Universidad Nacional Autónoma
De México

Facultad de Ciencias

Simulación interactiva de la tectónica de placas en la esfera terrestre

T E S I S

Que para obtener el título de:
Licenciado en Ciencias de la Computación

P R E S E N T A :

Félix Alejandro García Gutiérrez

Tutor: M. en C. Alejandro Aguilar Sierra

México, D.F. 2014

Resumen

El presente trabajo consta de la creación de una simulación interactiva que permite visualizar la configuración de los continentes por la Tectónica de Placas a través del tiempo, desde la Pangea hasta la época actual, si bien este es el rango que se tiene con los datos por defecto, esto puede cambiar dependiendo del rango de años de los que se tenga información y se carguen al programa al inicio de su ejecución.

Para lograr la simulación de la evolución tectónica se necesitaron los datos de las rotaciones de las placas tectónicas que abarcan el tiempo que se quiera visualizar, obteniendose como latitud, longitud y ángulo, convirtiendose a un formato binario el cuál el programa lee al inicio de la ejecución.

Además de los datos de las rotaciones se necesita la representación tridimensional de cada placa, contenidas dentro de un archivo de tipo .obj; esta información será la que se visualizará de forma dinámica e interactiva dentro del software resultante y como apoyo a la educación de las Ciencias de la Tierra en el tema de Tectónica de Placas.

Para comprender las rotaciones de las placas tectónicas se hace uso de las rotaciones de Euler y los cuaterniones, estos últimos son ampliamente usados hoy en día en el área de graficación por computadora y visualización.

Se cuenta con una serie de texturas que aportan información relevante sobre el tema y que son mapeadas a la representación del globo terraqueo y de las placas tectónicas.

Finalmente una interface gráfica simple pero intuitiva permite el manejo de este software.

Agradecimientos

A mi madre, mi tía María y mi hermanita María Elena por estar conmigo siempre guiándome, dándome consejos y enseñandome a ser mejor cada día, gracias por todo el apoyo, amor y comprensión que me han dado para poder llegar hasta aquí.

A mis tíos, primos y padrinos, por ilustrarme con sus historias, pláticas y diferentes maneras de ver la vida.

A ti Laura por estar ahí para vivir mis locuras y sueños, y ayudarme a verlos realizados.

Al M. en C. Alejandro Aguilar Sierra por el apoyo y paciencia en el largo proceso para la finalización de este trabajo y por ayudarme a obtener el tema de la presente tesis mediante el proyecto Ixtli del Centro de Ciencias de la Atmósfera con número IX100710.

Índice general

1. Marco Teórico **6**
- 1.1. Necesidad . . . 6
- 1.2. Estructura de la Tierra. . . . 7
  - 1.2.1. Composición Química. . . . 7
  - 1.2.2. Composición Física . . . 7
- 1.3. Tectónica de Placas. . . . 8
  - 1.3.1. Deriva Continental . . . 8
  - 1.3.2. Expansión del Fondo Oceánico . . . 11
  - 1.3.3. Movimiento de las Placas Tectónicas . . . 14
    - 1.3.3.1. Límites Divergentes . . . 14
    - 1.3.3.2. Límites Convergentes . . . 14
    - 1.3.3.3. Límites Transformantes . . . 16
- 1.4. Rotaciones de Placas y Polos de Euler . . . 16
  - 1.4.1. Rotaciones de Euler . . . 17
  - 1.4.2. Ecuaciones de las Rotaciones de Euler . . . 17
- 1.5. Cuaterniones . . . 19
  - 1.5.1. Cuaternión . . . 19
  - 1.5.2. Rotación de Cuaterniones . . . 21
  - 1.5.3. Forma Matricial . . . 24
  - 1.5.4. Interpolación Esférica Lineal . . . 25

2. Realidad Virtual **29**
- 2.1. ¿Qué es la Realidad Virtual? . . . 29
- 2.2. Inmersión . . . 30
- 2.3. Inmersión . . . 31
- 2.4. Realidad Virtual y sus aplicaciones en la enseñanza . . . 31

3. Preparación de Datos **32**
3.1. Datos de las Placas Tectónicas 32
3.2. Construcción de Mallas 3D a Partir de Polígonos 2D 35
3.3. Nomenclatura de las Placas Tectónicas 36

4. Desarrollo del Sistema **38**
4.1. Captura de Requerimientos 38
4.1.1. Diagrama de Casos de Uso 39
4.1.2. Análisis de Requerimientos 40
4.2. Diseño 41
4.2.1. Arquitectura de Software 42
4.2.2. Modelo Vista Controlador (MVC) 42
4.3. Diagrama de Paquetes 43

5. Construcción **45**
5.1. Introducción 45
5.1.1. OpenGL 48
5.1.2. OpenSceneGraph 49
5.1.2.1. Grafos de Escena 50
5.1.3. GLSL 50
5.2. Grafo General de Escena 51
5.3. Creación de la Tierra y Texturas 53
5.4. Barra de Edades y Eventos de Usuario 58
5.5. Selección e Información de Placas Tectónicas 71
5.6. Animación 72

6. Resultados **74**
6.1. Problemas: 74
6.2. Soluciones: 75

7. Conclusiones y trabajo futuro **79**

Capítulo 1

Marco Teórico

1.1. Necesidad

Este trabajo surge como parte del proyecto Ixtli para su aprovechamiento en la enseñanza del tema de Tectónica de Placas en las carreras de Ingeniería afines y Ciencias de la Tierra, ya sea en las instalaciones de la sala Ixtli o en las aulas donde se imparta este tema.

Debido a la importancia de la Tectónica de Placas para la comprensión del funcionamiento del planeta Tierra, es fundamental que los alumnos comprendan a profundidad este tema. La aportación de este trabajo es brindar una herramienta complementaria en su formación educativa, para la mejor comprensión del tema. Las ventajas obtenidas con un programa visual interactivo son:

- El estudiante tiene un mayor contacto con la información que puede modificar a su gusto y enfocarse en los aspectos que le resulten dificil de comprender.
- La interactividad permite la manipulación de la información, lo que ayuda a mejorar la retención de datos, ya que se está interactuando y no solo se está observando.

Para lograr esto se necesita como base los conocimientos de la Téctonica de Placas, dando como resultado que la simulación del movimiento sea lo más real posible y la aplicación de las técnicas de Realidad Virtual para que la experiencia de usuario sea enriquecedora.

Este capítulo pretende explicar los conocimientos teóricos que serán utilizados durante la creación del software.

1.2. Estructura de la Tierra.

La estructura de la Tierra está dividida en capas específicas, esta clasificación se puede hacer según dos criterios: la composición química y las propiedades físicas.

1.2.1. Composición Química.

Según la composición química existen 3 capas principales que componen la Tierra y que son nombradas empezando por la capa más superficial : *corteza, manto y núcleo.*

- La *corteza* es la capa externa que tiene de grosor entre 3 km en las dorsales oceánicas y 70 km en las grandes cordilleras terrestres.
- El *manto* cubre desde donde termina la corteza hasta una profundidad de 2890 km, lo cual la hace la capa más grande. Está compuesta por rocas silíceas que debido a las altas temperaturas hacen que las rocas puedan fluir, aunque de una manera muy lenta.
- El *núcleo* es la capa más profunda de la Tierra. Está compuesta por dos partes: la parte interna sólida de 1220km de radio y, una capa externa semisólida que tiene hasta 3400km. El núcleo está compuesto en su mayoria por hierro (80 %) , niquel y otros elementos ligeros.

1.2.2. Composición Física

La composición física define 4 capas: *litósfera, astenósfera, mesósfera y núcleo.*

- La *litósfera* es la capa sólida externa y más fría de la Tierra y tiene un grosor entre 100 km y 250 km, la cual "flota" sobre la astenósfera. La litósfera, está fragmentada en docenas de placas de varios tamaños que se mueven unas contra otras alrededor de toda la Tierra.

- La *astenósfera* es la capa que está por debajo de la litósfera y alcanza los 660 km de profundidad, en la parte superior de esta capa se encuentra una pequeña porción de roca fundida, lo cual provoca que la litósfera pueda moverse independientemente.
- La *mesósfera* es una capa más rígida y que se encuentra debajo de la astenósfera, además de ser muy caliente, su profundidad llega hasta los 2900km.
- El *núcleo* al igual que en el criterio por composición química se divide en dos: el núcleo externo cuyas corrientes de convección genera el campo magnético de la Tierra y el núcleo interno es una esfera de 1216 km de radio. Esta esfera tiene una temperatura muy elevada debido a la presión que se ejerce sobre ella.

1.3. Tectónica de Placas.

La teoría de la Tectónica de Placas dice que la *corteza* de la Tierra está fragmentada en un conjunto de *placas tectónicas* de varios tamaños. Una placa tectónica es una gran loza de roca sólida de forma irregular compuesta de partes continentales y oceánicas de la litósfera, su tamaño puede variar desde unos cientos hasta miles de kilómetros. La profundidad de estas placas varia desde menos de 15 km en partes donde existe litósfera de reciente creación, hasta aproximadamente 200 km o más en la litósfera continental.

Esta teoría ha sido formulada recientemente (siglo XX) y ha cambiado la forma en la que se entiende la dinámica de la Tierra y que los científicos se preguntaban hace siglos, con esto se pueden dar respuesta a varios fenómenos geológicos como la creación de las cordilleras o los sismos que suceden en diversas partes del mundo. Para poder llegar a este resultado se hizo uso de varias ramas de las ciencias de la Tierra como la paleontología y la sismología, además de dos teorías mas antiguas que fueron fundamentales para la creación de la Tectónica de Placas: la *Deriva Continental* y la *Expansión del Fondo Oceánico*.

1.3.1. Deriva Continental

Desde antes del siglo XX se ha sospechado que los continentes no han estado fijos a través de los años, dicha sospecha fue presentada en 1595 por

Abraham Ortelius, quien sugería que el conteniente americano se había separado de Europa y África, apoyándose en la forma en la que los límites costeros de los continentes coinciden. Esta idea quedó en el olvido hasta que resurgió como una teoría científica en 1912 cuando Alfred Lothar Wegener publicó dos artículos sobre una teoría llamada: *Deriva Contiental*. Esta teoría estaba basada en como Sudáfrica y Sudamérica encajaban si se unían las costas, trata sobre el desplazamiento lento y contínuo de las masas continentales y que se basaba en lo que Ortelius había observado en el pasado, aunado a esto se encontraban los descubrimientos del parecido de flora, fauna y fósiles hallados en las costas donde se suponía que los continentes habían estado unidos y ahora los separaba el oceáno atlántico, haciendo imposible que los organismos se hubieran trasladado nadando de una costa a otra, por lo cual Wegener decía que hace 250 millones de años había existido un único continente llamado *Pangea*.

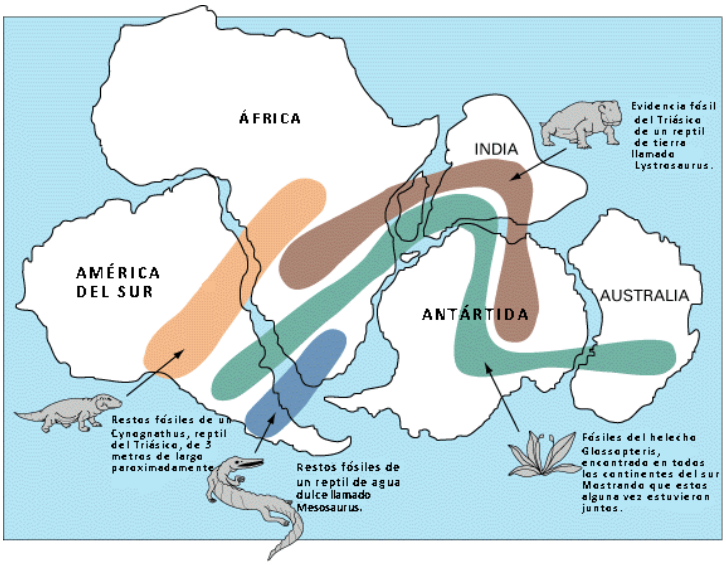

Figura 1.1: Evidencia de restos fósiles en varios lugar del mundo que hoy en dia se encuentran separados (Kious and Tilling [5]).

El nombre de *Pangea* viene del prefijo "pan" que significa todo y "gea" que significa suelo o tierra, por lo que Pangea significa: toda la tierra, hace 250 millones de años solo existía este supercontinente en la superficie de la Tierra, aunado a este supercontinente existía un megaoceáno llamado *Pantalasa*. Se

cree que la forma que tenía *Pangea* era en forma de C a la altura del ecuador, permitiendo que los animales existentes cruzaran de un hemisferio a otro por tierra, sin ningúna barrera acuática. La desintegración de la Pangea se realizó en varias fases.

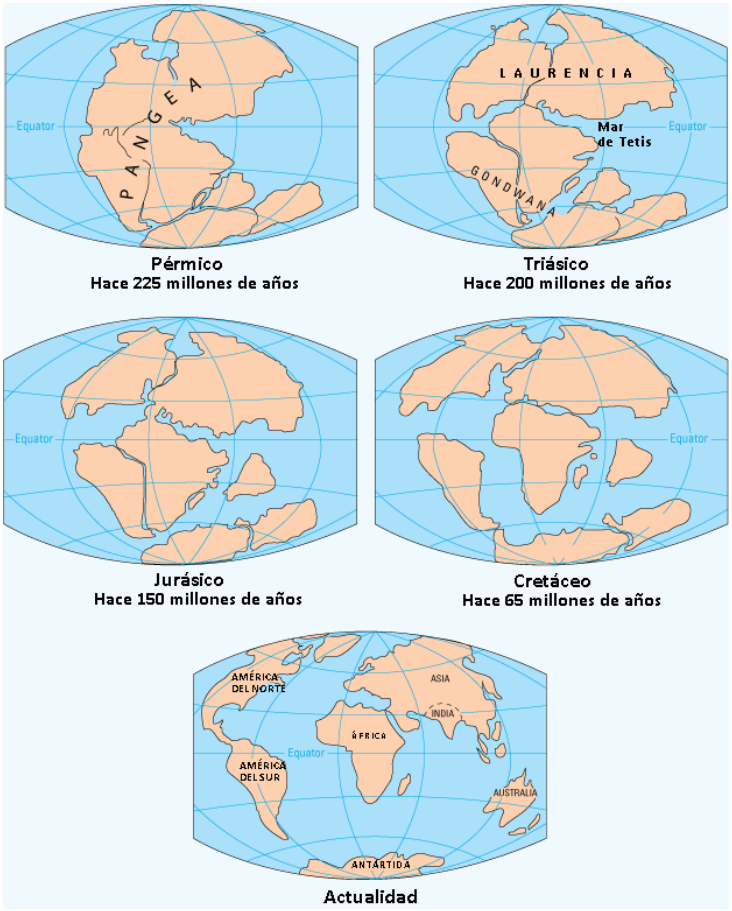

Figura 1.2: Pangea y su fragmentación a través de los años (Kious and Tilling [5]).

La primera fase ocurrió alrededor de 200 millones de años, el supercontinente *Pangea* comenzó a divirse en dos grandes masas continentales, una en el hemisferio norte llamada *Laurencia* y otra en el hemisferio sur llamada $++dwana$. Ésta separacion de Pangea provocó la creación del oceáno *Tetis*, el cuál ocupó el lugar entre las dos masas continentales. Además se crearon dos grietas, una en Laurencia que dio inicio a la separación entre Norteamérica

y África iniciando el oceáno Atlántico Norte, la otra grieta fue en Gondwana y separó a África de Australia y a la Antártida.

La segunda fase se llevó acabo hace 135 millones de años cuando el continente Gondwana inició la separación de África y Sudamérica, dando paso al oceáno Atlántico Sur, el territorio que hoy corresponde a la India tomó camino hacia Asia alejandose de África.

Hace 65 millones de años tuvo espacio la siguiente fase, en la cuál Madagascar se separa del África y el oceano Atlántico Sur se hace más grande, la India continua su viaje para encontrarse con Asia. Norteamérica sigue unido a Europa y Asia.

En la última fase aproximadamente hace 45 millones de años, la masa continental de Norteamérica y Groenlandia se separaron de Europa y Asia en el hemisferio norte, dando paso al Mar Noruego y expandiendo el oceáno Atlántico; el oceáno Índico se siguió expandiendo, Australia se separó de la Antartida y se movió hacia el norte. Mucho del panorama actual de los continentes comenzó en esta fase: Se crearon los Himalayas debido al choque de la Índia con Asia, el levantamiento de los Alpes, la apertura del Mar de Japón y la formación de Baja California.

Aún con toda esta evidencia, esta teoría tenía una gran debilidad: ¿Qué clase de fuerza era la que hacia posible el movimiento de las gigantescas masas continentales?, Wegener argumentó que era debido a que las masas continentales flotaban sobre la corteza oceánica lo que permitia su movimiento, esto fue refutado por sus críticos argumentando que era imposible mover tales masas continentales sin que estas se rompieran en el proceso.

1.3.2. Expansión del Fondo Oceánico

Después de que la teoría de la *Deriva Continental* fuera debatida por mucho tiempo y con fallas en puntos importantes en su proposición, al inicio de la década de los cincuentas, varias evidencias sobre el comportamiento de la Tierra volvieron a poner el debate sobre la mesa, esto último llevaria a la teoría de la *Expansión del Fondo Oceánico*, la cual fue posible por el avance de los aparatos disponibles para medir la profundidad del oceano, permitiendo a los científicos tener una mejor y mas precisa manera de analizar el piso marino.

El primer indicio de la nueva teoría surgió al estar haciendo pruebas sobre el fondo del oceáno, se encontró que el piso era mucho mas delgado de lo que se pensaba y que existía una gran cadena montañosa que le daba vuelta a

la Tierra, la cadena montañosa se encontraba en el fondo del oceáno y fue llamada cordillera meso-oceánica.

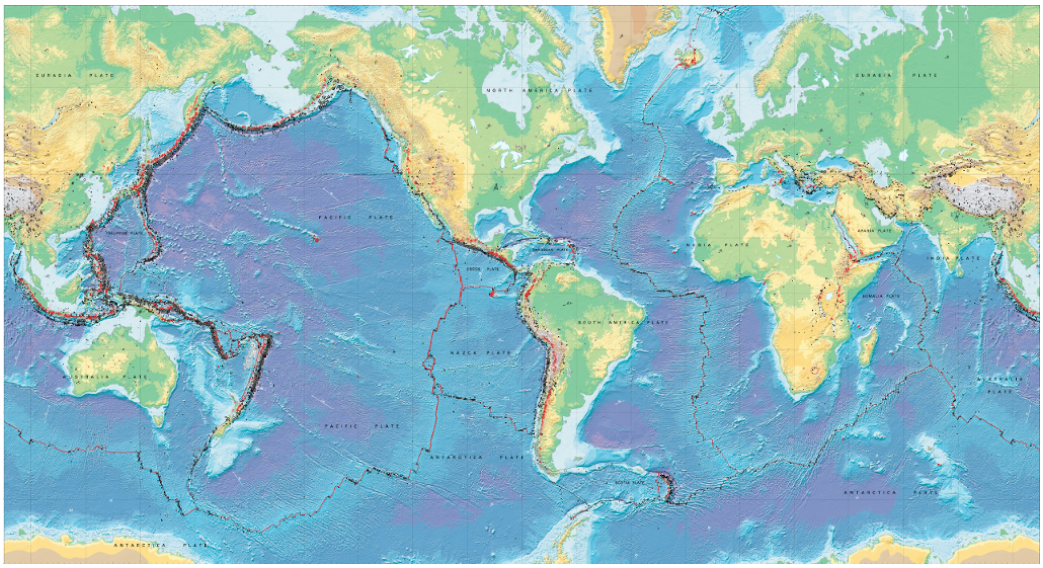

Figura 1.3: cadena montañosa a lo largo del fondo oceánico (usgs.gov)

Otro elemento importante para esta teoría fue el descubrimiento de las *Anomalías Magnéticas* encontradas en el fondo del oceáno, para ello se usaron magnetómetros que permitieron reconocer la polaridad de la roca basáltica, encontrándose variaciones magnéticas, ya que en algunas rocas tenían como Norte magnético el polo Sur de la Tierra, la respuesta a este comportamiento es que la magnetita presente dentro del magma se ajusta al campo magnético de la Tierra, y al convertirse en roca volcánica solidificada se queda con la configuración magnética que tenía la Tierra. Por medio de otras investigaciones se sabe que la polaridad del campo geomagnetico terrestre cambia cada cierto tiempo. Además estas anomalías magnéticas estaban alineadas simétricamente a ambos lados de la cordillera-meso oceánica.

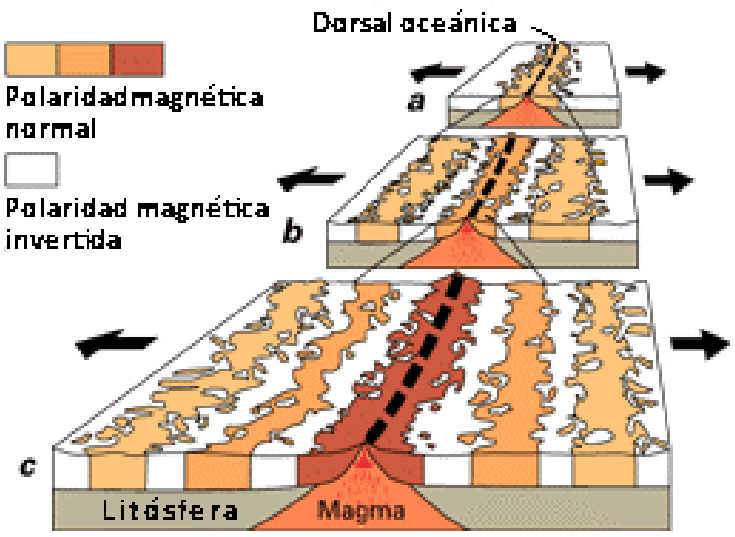

Figura 1.4: Variaciones de polaridad magnética en la roca basáltica(Kious and Tilling [5]).

Esta teoría tomó todos los elementos anteriores para dar las bases de cual es la fuerza que actúa en la superficie de la tierra y hace que los continentes se desplacen, y el cuál era el punto débil de la *Deriva Continental* para darla por válida.

La *Expansión del Fondo Oceánico* sostiene que en las zonas más delgadas del océano, se encuentran las dorsales océanicas, por cuyo centro sale magma que divide al piso en dos partes. Cada parte está a un lado de la dorsal, el magma al enfriarse, crea nuevo piso marino, el cual registra la polaridad que tenga el campo magnético de la Tierra en ese momento. Todo esto da como resultado que el nuevo piso empuje al viejo y lo vaya alejando de la dorsal, con lo cual se complementa la teoría de la *Deriva Continental*.

Debido al movimiento de *Expansión del Fondo Oceánico* y a que el mundo es esférico y finito, los objetos que se separan debido a las dorsales oceánicas se volverán a juntar e interactuar con los demás objetos que se encuentren en la superficie y que se muevan más lento, más rápido o en diferentes direcciones que el resto. La forma en la que se mueven estos objetos y las consecuencias de la interacción de estos objetos se explica a continuación.

1.3.3. Movimiento de las Placas Tectónicas

La unificación de las teorías anteriores dio como resultado la *Tectónica de Placas,* que describe los eventos que suceden en la litósfera, y la interacción entre las placas que la conforman. Dichas interacciones se dan en los límites de una placa con otra. Dependiendo de cómo se comporten las placas en esos lugares, los límites se clasifican en: divergentes, convergentes y transformantes.

1.3.3.1. Límites Divergentes

Son los límites en donde nace nuevo magma debido a la actividad volcánica en estas zonas y por lo tanto las placas se separan una de otra en un proceso lento. El mejor ejemplo de este tipo de límites es el que se encuentra enmedio del oceáno Atlántico y que se encuentra conformado por las placas de Eurasia con Norteamérica y las de África con Sudamérica.

1.3.3.2. Límites Convergentes

Son aquellos límites en donde las placas chocan unas con otras, generando alguno de los eventos siguientes:

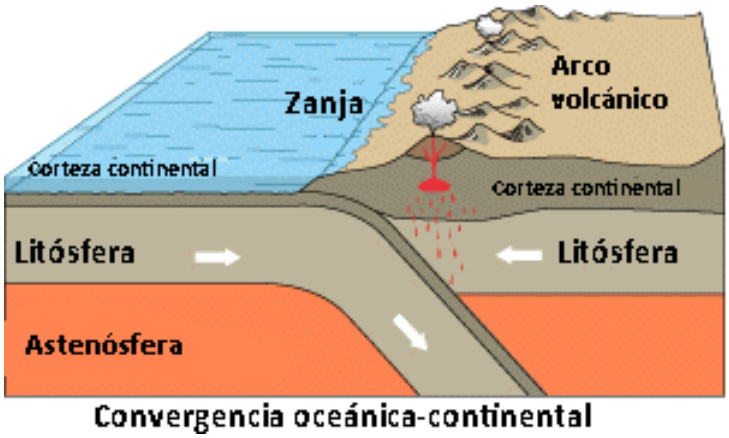

Figura 1.5: Convergencia de una placa continental con una placa oceánica (Kious and Tilling [5]).

Una placa es subducida bajo otra placa, cuando una placa continental choca con una placa oceánica esta última es la que sera subducida, y en la

superficie se crean cadenas volcánicas.

Figura 1.6: Convergencia de dos placas oceánicas (Kious and Tilling [5]).

Cuando el choque es entre dos placas oceánicas una se subduce y esto da origen a volcanes marinos que pueden dar lugar a cadenas de islas volcánicas, denominadas arcos de islas.

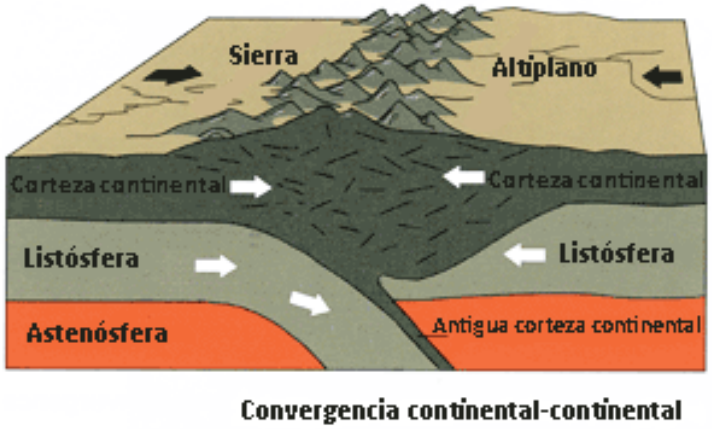

Figura 1.7: Convergencia de dos placas continenales (Kious and Tilling [5]).

Al chocar dos placas continentales ninguna es subducida, en cambio estas placas son empujadas hacia arriba, creando cadenas montañosas de gran elevación como las del Himalaya.

1.3.3.3. Límites Transformantes

También conocidos como *límites de falla de transformación*, en estas zonas las placas tectónicas no convergen o divergen como se ha explicado anteriormente, sino que estan conectadas dos dorsales oceánicas creando un desplazamiento horizontal, donde las placas se desplazan una a lado de la otra creando márgenes en forma de zig-zag, usualmente no causan demasiados daños cuando se encuentran en el oceáno. Un ejemplo de donde se encuentra este tipo de límite es la falla de San Andrés.

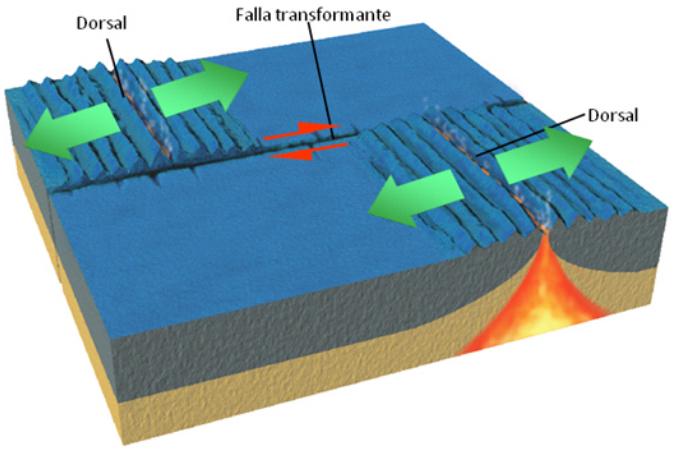

Figura 1.8: Límites transformantes

1.4. Rotaciones de Placas y Polos de Euler

Las rotaciones de Euler describen los movimientos de placas rígidas sobre una esfera y pueden ser usadas para realizar reconstrucciones paleográficas y comparar datos paleomagnéticos de diferentes continentes.

La reconstrucción de las posiciones de los continentes es el mayor propósito de la tectónica de placas. Sin embargo hasta hace muy poco tiempo, la reconstrucción se hacia cortando y reacomodando continentes sobre un mapa, con lo cual al ser una proyeccion bidimensional de una esfera terrestre tridimiensional, lo que provoca que los contornos de los continentes luzcan de manera diferente en el mapa, de como son originalmente en la esfera terrestre. Para poder obtener la mayor fidelidad posible es indispensable realizar

las reconstrucciones en proyecciones tridimiensionales.

1.4.1. Rotaciones de Euler

Los movimientos de las placas rígidas en la superficie terrestre pueden ser descritos por rotaciones de Euler, dichas rotaciones son realizadas alrededor de un eje de rotación que pasa por el centro de la Tierra. La intersección del eje de rotación con la superficie de la Tierra se le llama *polo de euler*. Los fundamentos matemáticos para las rotaciones en una esfera están dados por el Teorema de Euler, el cual dice que el movimiento general de un cuerpo rígido con un punto fijo puede ser descrito por una rotación[2]; lo cual nos lleva a que el movimiento de las placas rígidas sobre la Tierra puede ser descrito por rotaciones.

1.4.2. Ecuaciones de las Rotaciones de Euler

[2]Un punto P en la superficie terrestre está definido por su latitud P_λ y su longitud P_φ o por sus coordenadas Cartesianas P_x, P_y, P_z entonces:

$$P = (P_\lambda, P_\varphi) = (P_{x,} P_y P_z) \tag{1.1}$$

tal que:

$$\begin{aligned} P_x &= Rcos(P_\lambda)cos(P_\varphi) \\ P_y &= Rcos(P_\lambda)sin(P_\varphi) \\ P_z &= Rsin(P_\lambda) \end{aligned} \tag{1.2}$$

Donde R es el radio de la Esfera Terrestre, y comúnmente para simplificar se toma con el valor unitario.

Una rotacion sobre una esfera está definida por un polo de Euler: $\overrightarrow{E}$ y un ángulo de Euler: ϕ, el cual es medido en sentido contrario de las manecillas del reloj. Estos dos parámetros definen la rotación de Euler:

$$Rotacion\ de\ Euler = ROT[\overrightarrow{E}, \phi]. \tag{1.3}$$

Un punto $\overrightarrow{P}$ puede ser transformado con una rotación de Euler al punto $\overrightarrow{P}'$, lo cual se hace multiplicando una matriz de rotación A con el vector

$$\overrightarrow{P} = A\overrightarrow{P}' \tag{1.4}$$

Para poder derivar la matriz de rotación A, la rotación alrededor del polo de Euler es dividida en 3 rotaciones.

1. Una transformación T en el sistema de coordenadas donde $\overrightarrow{E}$ es el vector unidad sobre el eje Z.

2. Una rotación R sobre el eje Z con el algulo de rotación ϕ.

3. Una transformación T^{-1}que regresa a las coordenadas originales.

Los elementos de la matriz T son cosenos directores, cosenos de los ángulos entre los ejes del primer sistema de coordenadas(x,y,z) y el sistema de coordenadas transformado(x',y',z'):

$$T = \begin{pmatrix} cos(x,x') & cos(y,x') & cos(z,x') \\ cos(x,y') & cos(y,y') & cos(z,y') \\ cos(x,z') & cos(y,z') & cos(z,z') \end{pmatrix} \tag{1.5}$$

Si estos elementos son derivados de la latitud E_λ y la longitud E_φ del polo de Euler, entonces:

$$T = \begin{pmatrix} sinE_\lambda cosE_\varphi & sinE_\lambda sinE_\varphi & -cosE_\lambda \\ -sinE_\varphi & cosE_\varphi & 0 \\ cosE_\lambda cosE_\varphi & cosE_\lambda sinE_\varphi & sinE_\lambda \end{pmatrix} \tag{1.6}$$

Por otra parte tenemos que la rotación R sobre el polo de Euler transformado con el ángulo ϕ es una rotación sobre el eje Z:

$$R = \begin{pmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.7}$$

Dado que la matriz T es una transformación de una base ortonormal, T es una matriz ortogonal, su inversa es igual a su transpuesta.

Al final de las 3 transformaciones nos queda:

$$A = T^{-1}RT \tag{1.8}$$

multiplicando los elementos tenemos:

$$A = \begin{pmatrix} E_x^2(1-cos\phi)+cos\phi & E_xE_y(1-cos\phi)-E_zsin\phi & E_xE_z(1-cos\phi)+E_ysin\phi \\ E_yE_x(1-cos\phi)+E_zsin\phi & E_y^2(1-cos\phi)+cos\phi & E_yE_z(1-cos\phi)-E_xsin\phi \\ E_zE_x(1-cos\phi)-E_ysin\phi & E_zE_y(1-cos\phi)+E_xsin\phi & E_z^2(1-cos\phi)+cos\phi \end{pmatrix} \tag{1.9}$$

Esta matriz será la que utilicemos para realizar los movimientos de las placas tectónicas.

1.5. Cuaterniones

Un cuaternión es una entidad matemática comúnmente usada para representar una rotación en los programas de gráficos 3D. El uso de los cuaterniones sobre las matrices de rotación es debido a que estos ocupan menos espacio para guardar la información de las rotaciones. Se requieren menos operaciones aritméticas y además se pueden hacer interpolaciones de una manera más sencilla para producir animaciones[6].

1.5.1. Cuaternión

El conjunto de cuaterniones puede ser pensado como un espacio vectorial de cuatro dimensiones en el cual el elemento **q** :

$$q = \langle w, x, y, z \rangle = w + xi + yi + zk \tag{1.10}$$

Otra forma en la que comúnmente se escribe un cuaternión es $q = s + v$, donde s representa al escalar correspondiente de la componente **w** y v representa la parte vectorial que comprende a las componentes x, y, z de **q**.

El conjunto de los cuaterniones es una extensión de los números complejos, por lo cual la multiplicación esta definida por la ley de distributividad de los complejos más las reglas para la multiplicación de las componentes "imaginarias" i,j,k.

$$\begin{aligned} i^2 = j^2 = k^2 = -1 \\ ij = -ji = k \\ jk = -kj = i \\ ki = -ik = j \end{aligned} \tag{1.11}$$

Dadas estas reglas, para dos cuaterniones $q_1 = w_1 + x_1 i + y_1 j + z_1 k$, $q_2 = w_2 + x_2 i + y_2 j + z_2 k$ el producto $q_1 q_2$ queda como:

$$\begin{array}{c} q_1 q_2 = (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) \\ +(w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2) i \\ +(w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2) j \\ +(w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2) k \end{array} \tag{1.12}$$

Se debe tener cuidado al multiplicar cuaterniones ya que la multiplicacion no es conmutativa.

En el caso de que los cuaterniones estén representados de la forma escalar, vector, la multiplicación sería la siguiente:

$$q_1 q_2 = s_1 s_2 - v_1 \cdot v_2 + s_1 v_2 + s_2 v_1 + v_1 \times v_2 \tag{1.13}$$

La magnitud de un cuaternión esta definida como:

$$\|q\| = \sqrt{w^2 + x^2 + y^2 + z^2} \tag{1.14}$$

Al igual que los números complejos , los cuaterniones tienen conjugados, los cuales serán de gran ayuda mas adelante. El conjugado de un cuaternión se define como $\overline{q} = s - v$. Dada la definición del conjugado de q, se tiene la siguiente igualdad:

$$p\bar{p} = \bar{p}p = q \cdot q = \|q\|^2 \tag{1.15}$$

Para demostrar esto se utiliza la multiplicación 1.13. Sean $q = w + xi + yj + zk$ y $\bar{q} = w - xi - yj - zk$ tenemos que:

$$\begin{array}{c} q\bar{q} = (ww - x(-x) - y(-y) - z(-z)) + (w(-x) + xw + y(-z) - z(-y))i \\ +(w(-y) - x(-z) + yw + z(-x))j + (w(-z) + x(-y) - y(-x) + zw)k \\ = ww + xx + yy + zz + 0i + 0j + 0k \\ = w^2 + x^2 + y^2 + z^2 \\ \bar{q}q = (ww + xx + yy + zz) + (wx - xw - yz - zy)i \\ +(wy + xz + -yw - zx)j + (wz - xy + yx - zw)k \\ = ww + xx + yy + zz + 0i + 0j + 0k \\ = w^2 + x^2 + y^2 + z^2 \\ q \cdot q = ww + xx + yy + zz = w^2 + x^2 + y^2 + z^2 \\ \|q\|^2 = \sqrt{ww + xx + yy + zz}^2 = w^2 + x^2 + y^2 + z^2 \end{array}$$

Lo cual demuestra que la igualdad es cierta ya que todas las operaciones dan el mismo resultado.

El inverso de un cuaternión distinto de cero **q** y que denotaremos como q^{-1} está dado como:

$$q^{-1} = \frac{\bar{q}}{\|q\|^2} \tag{1.16}$$

Para demostrar que esto es cierto utilizaremos la igualdad 1.15 en la página anterior para demostrar que $qq^{-1} = 1$y $q^{-1}q = 1$

$$qq^{-1} = \frac{q\bar{q}}{\|q\|^2} = \frac{\|q\|^2}{\|q\|^2} = 1 \tag{1.17}$$

$$q^{-1}q = \frac{\bar{q}q}{\|q\|^2} = \frac{\|q\|^2}{\|q\|^2} = 1 \tag{1.18}$$

1.5.2. Rotación de Cuaterniones

Una rotación en tres dimensiones usando cuaterniones puede ser pensada como una función de rotación φ que mapea $\mathbb{R}^3$ en sí mismo. Dicha función debe preservar longitudes, ángulos y orientación de un punto P.

Para que la longitud se preserve se necesita que las magnitudes sean las mismas:

$$\|\varphi(P)\| = \|P\| \tag{1.19}$$

El ángulo formado entre dos puntos P_1 y P_2 se conserva si el producto punto se conserva una vez aplicada la función:

$$\varphi(P_1) \cdot \varphi(P_2) = P_1 \cdot P_2 \tag{1.20}$$

La orientación de la rotación se mantiene si el vector normal mantiene la misma dirección:

$$\varphi(P_1) \times \varphi(P_2) = \varphi(P_1 \times P_2) \tag{1.21}$$

Para probar esto se necesita que la función φ se extienda a un mapeo de un espacio $\mathbb{H}$ sobre si mismo requiriendo que $\varphi(s + v) = s + \varphi(v)$, lo cual permite reescribir la ecuación 1.20:

$$\varphi(P_1) \cdot \varphi(P_2) = \varphi(P_1 \cdot P_2) \tag{1.22}$$

Asumiendo que P_1 y P_2 como cuaterniones con parte escalar cero, permite combinar las ecuaciones de preservación de la orientación 1.21 y ángulo 1.22 ya que el producto de los cuaterniones 1.13 con escalar cero se puede ver como $P_1P_2 = -P_1 \cdot P_2 + P_1 \times P_2$, por lo tanto la unión de las ecuaciones de angulo y preservación queda como

$$\varphi(P_1)\varphi(P_2) = \varphi(P_1P_2) \tag{1.23}$$

Una función φ que satisface esta ecuación tambien se le llama homomorfismo.

Una clase de funciones dadas por:

$$\varphi_q(P) = qPq^{-1} \tag{1.24}$$

Donde q es un cuaternión distinto de cero, satistace las ecuaciones 1.19 y 1.23. Primero veremos que preserva la longitud:

$$\begin{gathered} \|\varphi_q(P)\| = \|qPq^{-1}\| = \|q\| \, \|P\| \, \|q^{-1}\| \\ utilizando\ 1.16 \\ = \|P\| \, \|q\| \left\| \tfrac{\bar{q}}{\|q\|^2} \right\| = \|P\| \left\| \tfrac{q\bar{q}}{\|q\|^2} \right\| = \|P\| . \end{gathered} \tag{1.25}$$

Ahora veamos que preserva la orientación y el ángulo:

$$\varphi_q(P_1)\varphi_q(P_2) = qP_1q^{-1}qP_2q^{-1} = qP_1P_2q^{-1} = \varphi_q(P_1P_2) \tag{1.26}$$

Ahora que se comprobó que esta función cumple con las preservaciones, se necesita encontrar una fórmula para que el cuaternión q corresponda a una rotación con ángulo θ sobre un eje dado A.

Sea $q = s + v$ un cuaternión unidad, entonces $q^{-1} = s - v$ y dado un punto P tenemos:

$$\begin{aligned} &qPq^{-1} = (s+v)P(s-v) \\ &= (-v \cdot P + sP + v \times P)(s-v) \\ &= -sv \cdot P + s^2P + sv \times P + (v \cdot P)v - sPv - (v \times P)v \\ &= s^2P + 2sv \times P + (v \cdot P)v - v \times P \times v. \end{aligned} \tag{1.27}$$

Aplicando la igualdad al producto cruz $P \times (Q \times P) = P \times Q \times P = P^2Q - (P \cdot Q)P$ tenemos:

$$qPq^{-1} = (s^2 - v^2)P + 2sv \times P + 2(v \cdot P)v \tag{1.28}$$

Poniendo $v = tA$ donde A es un vector unitario, la ecuación puede ser reescrita como:

$$qPq^{-1} = (s^2 - t^2)P + 2stA \times P + 2t^2(A \cdot P)A \tag{1.29}$$

Comparando esta ecuación con la ecuación sobre una rotación sobre un eje cualquiera $P' = Pcos\theta + (A \times P)sin\theta + A(A \cdot P)(1 - cos\theta)$ se pueden inferir las siguientes igualdades:

$$\begin{aligned} s^2 - t^2 &= cos\theta \\ 2st &= sin\theta \\ 2t^2 &= 1 - cos\theta \end{aligned} \tag{1.30}$$

De la tercera igualdad podemos obtener:

$$t = \sqrt{\frac{1 - cos\theta}{2}} = sin\frac{\theta}{2} \tag{1.31}$$

de la primera y tercera igualdad obtenemos $s^2 + t^2 = 1$ ya que:

$$\begin{aligned} &s^2 - t^2 = cos\theta \\ &s^2 - t^2 + t^2 - t^2 = cos\theta \\ &s^2 - 2t^2 + t^2 = cos\theta \\ &s^2 - 1 + cos\theta + t^2 = cos\theta \\ &s^2 + t^2 = 1 \end{aligned}$$

este último resultado nos da $s = \sqrt{\frac{1+cos\theta}{2}} = cos(\theta/2)$con lo cual se puede verificar la segunda igualdad. Con estos resultados se determina que el cuaternión unidad q que realiza una rotación de un ángulo θ sobre un eje A está dado por:

$$q = cos\frac{\theta}{2} + Asin\frac{\theta}{2} \tag{1.32}$$

Si multiplicamos un escalar por el cuaternion q, este sigue representando la misma rotación, esto se puede ver si a es un escalar y:

$$(aq)P(aq)^{-1} = aqP\frac{q^{-1}}{a} = qPq^{-1} \tag{1.33}$$

Otro punto importante es que el producto de dos cuaterniones también representan una rotación. Sean q_1 y q_2 dos cuaterniones, el resultado de la multiplicación es la rotación de q_2 y despues de q_1:

$$q_1(q_2Pq_2^{-1})q_1^{-1} = (q_1q_2)P(q_1q_2)^{-1} \tag{1.34}$$

Esto permite realizar cualquier número de rotaciones y el resultado sera un cuaternión que contenga todas las rotaciones.

Los cuaterniones son muy usados en temas de computación gráfica y videojuegos, ya que requiere menos operaciones de suma y multiplicacion para resolver la multiplicación de cuaterniones en comparación con la multiplicación de matrices de 3x3, pues el primero requiere 16 operaciones mientras que el segundo 27. Dicho número de operaciones es significativo al tener que hacer varias rotaciones sobre un objeto o múltiples rotaciones entre múltiples objetos en el menor tiempo posible.

1.5.3. Forma Matricial

A veces es necesario convertir un cuaternión a una matriz de rotación de 3x3 para que la transformación de un objeto se pueda efectuar en una biblioteca gráfica 3D. Para poder determinar la representación matricial correspondiente al cuaternión $q = s+tA$ escribiremos la ecuación 1.29 en tforma matricial para esto tomaremos en cuenta que si P y Q son 2 vectores:

$$(P \cdot Q)Q = \begin{bmatrix} Q_x^2 & Q_xQ_y & Q_xQ_z \\ Q_xQ_y & Q_y^2 & Q_yQ_z \\ Q_xQ_z & Q_yQ_z & Q_z^2 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \tag{1.35}$$

y

$$P \times Q = \begin{bmatrix} 0 & -P_z & P_y \\ P_z & 0 & -P_x \\ -P_y & P_x & 0 \end{bmatrix} \begin{bmatrix} Q_x \\ Q_y \\ Q_z \end{bmatrix} \tag{1.36}$$

Entonces reescribiendo la ecuación en forma matricial tendremos:

$$qPq^{-1} = \begin{bmatrix} s^2-t^2 & 0 & 0 \\ 0 & s^2-t^2 & 0 \\ 0 & 0 & s^2-t^2 \end{bmatrix} P + \begin{bmatrix} 0 & -2stA_z & 2stA_y \\ 2stA_z & 0 & -2stA_x \\ -2stA_y & 2stA_x & 0 \end{bmatrix} P + \begin{bmatrix} 2t^2A_x^2 & 2t^2A_xA_y & 2t^2A_xA_z \\ 2t^2A_xA_y & 2t^2A_xA_y2t^2A_y & 2t^2A_yA_z \\ 2t^2A_xA_z & 2t^2A_yA_z & 2t^2A_z^2 \end{bmatrix} P \tag{1.37}$$

escribiendo a **q** como $w = s, x = tA_x, y = tA_y, z = tA_z$ y dado que A es un vector unidad tenemos: $x^2 + y^2 + z^2 = t^2A^2 = t^2$, reescribimos la ecuación anterior con estas igualdades y nos da:

$$qPq^{-1} = \begin{bmatrix} w^2-x^2-y^2-z^2 & 0 & 0 \\ 0 & w^2-x^2-y^2-z^2 & 0 \\ 0 & 0 & w^2-x^2-y^2-z^2 \end{bmatrix} P + \begin{bmatrix} 0 & -2wz & 2wy \\ 2wz & 0 & -2wx \\ -2wy & 2wx & 0 \end{bmatrix} P + \begin{bmatrix} 2x^2 & 2xy & 2xz \\ 2xy & 2y^2 & 2yz \\ 2xz & 2yz & 2z^2 \end{bmatrix} P \tag{1.38}$$

Como **q** es un cuaternión unidad (ya que es una propiedad que se pidió para llegar a la ecuacion 1.29) tenemos que $w^2 + x^2 + y^2 + z^2 = 1$ esto lo podemos escribir como $w^2 - x^2 - y^2 - z^2 = 1 - 2x^2 - 2y^2 - 2z^2$ lo cual sustituyéndolo en las matrices que se han hecho y sumándolas tenemos la formula R_q que es la matriz de rotación correspondiente al cuaternión q:

$$R_q = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix} \tag{1.39}$$

1.5.4. Interpolación Esférica Lineal

Dado que los cuaterniones pueden ser representados por vectores, resultan útiles para realizar interpolaciones. Las interpolaciones son usadas cuando un objeto se quiere animar y se quiere tener un movimiento suave en todos los pasos de animación aunque solo se cuente con datos para algunos pasos.

La interpolación más simple es la *interpolación lineal.* Dados dos cuaterniones unitarios q_1q_2 el cuaternión linealmente interpolado esta dado por:

$$q(t) = (1-t)q_1 + tq_2 \tag{1.40}$$

La función $q(t)$ cambia suavemente de posición del cuaternión q_1 a q_2 mientras t varía de 0 a 1. Esta primera aproximación no mantiene la longitud unitaria del cuaternión $q(t)$ mientras varia en el tiempo por lo cual se normaliza la función:

$$q(t) = \frac{(1-t)q_1 + tq_2}{\|(1-t)q_1 + tq_2\|} \tag{1.41}$$

Con esto se tiene una función que traza el arco entre los dos cuaterniones.

Aunque la interpolación lineal es eficiente, tiene el incoveniente de que el movimiento de la función $q(t)$ no es constante. Al graficar la tasa de cambio del ángulo entre $q(t)$ y q_1 como $cos^{-1}(q(t) \cdot q_1)$ se observa que el movimiento es relativamente lento cerca del punto de inicio y punto final, y el movimiento se incrementa cuando $t = 1/2$.

Por estos detalles es necesario encontrar una función que interpole dos cuaterniones preservando la longitud unitaria y que se mueva a función constante en todos los tramos de la interpolación. Si q_1 y q_2 estan separados por un ángulo θ, entonces la función que se busca deberá de generar un ángulo θt entre $q(t)$ y q_1 mientras t varia de 0 a 1.

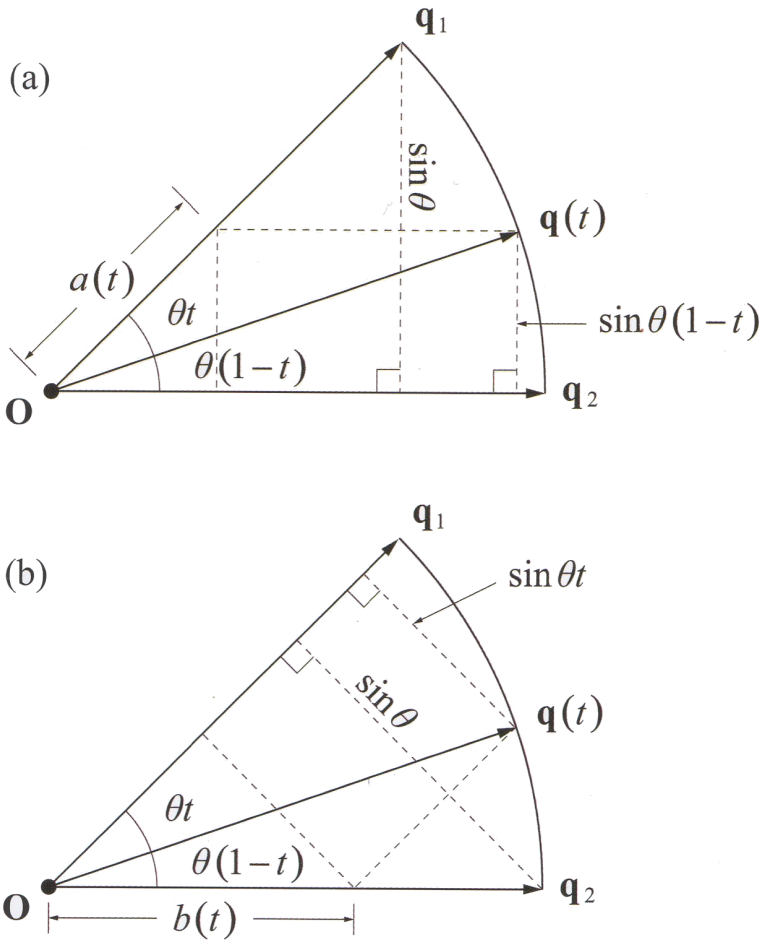

Figura 1.9: Triángulos semejantes formados por la traslación de la interpolación esférica (Lengyel [6])

De la figura anterior se puede observar que el cuaternión $q(t)$ está sobre el arco que conecta q_1q_2, formando el ángulo θt respecto a q_1 , para q_2 se forma el ángulo $\theta(1-t)$. entonces reescribimos la función $q(t)$ como:

$$q(t) = a(t)q_1 + b(t)q_2 \tag{1.42}$$

Tomando las funciones $a(t)$ y $b(t)$ como la representación de las longitudes de las componentes de $q(t)$ para la dirección de q_1 y q_2 respectivamente, se

puede determinar la longitud de $a(t)$ construyendo triángulos semejantes. La distancia perpendicular de q_1 al segmento de línea que conecta el origen con q_2 es igual a $\|q_1\| \, sin\theta$. La distancia perpendicular de $q(t)$ a la línea del origen a q_2 es $\|q(t)\| \, sin\theta(1-t)$ usando semejanza de triángulos, tenemos la relación:

$$\frac{a(t)}{\|q_1\|} = \frac{\|q(t)\| \, sin\theta(1-t)}{\|q_1\| \, sin\theta} \tag{1.43}$$

dado que los cuaterniones son de longitud unitaria tenemos $\|q_1\| = 1$ y $\|q(t)\| = 1$ por lo cual la ecuación queda como:

$$a(t) = \frac{sin\theta(1-t)}{sin\theta} \tag{1.44}$$

de la misma forma podemos crear las semejanzas del triángulo para $b(t)$ y tomando las mismas consideraciones de longitud para q_2 nos da la siguiente ecuación:

$$b(t) = \frac{sin\theta t}{sin\theta} \tag{1.45}$$

entonces sustituyendo en 1.42 tenemos nuestra *función de interpolación lineal* :

$$q(t) = \frac{sin\theta(1-t)}{sin\theta} q_1 + \frac{sin\theta t}{sin\theta} q_2 \tag{1.46}$$

y el ángulo está dado por

$$\theta = cos^{-1}(q_1 \cdot q_2) \tag{1.47}$$

además podemos reemplazar $sin\theta$ por:

$$sin\theta = \sqrt{1 - (q_1 \cdot q_2)^2} \tag{1.48}$$

reemplazando 1.48 dentro de 1.46 tenemos:

$$q(t) = \frac{\sqrt{1 - (q_1 \cdot q_2)^2} * (1-t)}{\sqrt{1 - (q_1 \cdot q_2)^2}} q_1 + \frac{sin\theta = \sqrt{1 - (q_1 \cdot q_2)^2} * t}{sin\theta = \sqrt{1 - (q_1 \cdot q_2)^2}} q_2 \tag{1.49}$$

permitiendo simplicar las operaciones necesarias para el cáculo del cuaternión.

Capítulo 2

Realidad Virtual

2.1. ¿Qué es la Realidad Virtual?

La realidad virtual en un principio se definió como sistemas tecnológicos capaces de crear mundos virtuales indistinguibles del mundo real (Gutiérrez et al. [3]); debido a restricciones tecnológicas esa meta en su sentido más estricto no ha sido posible de alcanzar. Lo que se busca es crear representaciones aceptables de la realidad con varios propósitos. La realidad virtual usa gráficos por computadora para crear ambientes 3D en los que las personas que usen dicho sistema puedan navegar para realizar determinadas interacciones dentro del ambiente virtual, para lo cual se usan dispositivos de hardware que complementen la experiencia como:

- HMD (Head Mounted Display): dispositivo que se coloca en la cabeza con unas pantallas a la altura de los ojos que despliegan el mundo virtual.
- Lentes estereoscópicos: Son lentes que en conjunto con pantallas especiales pueden simular la profundidad en los mundos virtuales.
- Guantes: Son usados en las manos y permiten tener un cursor virtual en el mundo para realizar interacciones, como abrir puertas.

Desarrollar sistemas de realidad virtual involucra diferentes disciplinas que estan ligadas con los sentidos humanos:

- Vista: Gráficos por computadora

- Oido: Sonido en 3D
- Tacto: Periféricos hápticos

Aunque el olfato y el gusto también son parte importante, debido a su complejidad para poder ser implementados, éstos no se incluyen dentro de los sistemas virtuales.

Además de los sentidos hay dos factores importantes que tienen que lograrse en un sistema virtual para que la experiencia sea lo más completa posible: inmersión y prescencia.

2.2. Inmersión

La inmersión esta relacionada con la interface física que el usuario tiene con la aplicación de realidad virtual, ésta puede ser clasificada en 3 tipos:

- *Sistema totalmente inmersivo*: La idea de estos sistemas es la de aislar completamente al usuario del mundo real con el fin de que el mundo virtual sea lo más creíble posible, un ejemplo del tipo de dispositivos que se usan en estos sistemas sería el HDM que encierra la vista en el mundo virtual, aunque sea un poco difícil moverse con tal dispositivo cargando en la cabeza.
- *Sistema semi inmersivo*: Un ejemplo de este tipo de sistemas son los CAVE (cueva), estos sistemas se componen de varias pantallas alrededor del usuario y que despliegan el mundo virtual de acuerdo a la posición del usuario dentro del mundo virtual, de ahí su nombre, ya que es introducir al usuario en una cueva donde se le mostrará el sistema virtual.
- *Sistema no inmersivo*: También llamados sistemas de realidad virtual de escritorio, éstos son desplegados en una monitor normal, estos sistemas se apoyan fuertemente en una gran interactividad, facilidad de uso, una gran calidad visual y auditiva, que logran captar totalmente la atención del usuario, un buen ejemplo de estos sistemas son los videojuegos.

2.3. Inmersión

La inmersión se refiere al estado psicológico en el que el usuario se siente dentro del mundo virtual, la inmersión se logra cuando la combinación de imagenes, sonidos y retroalimentación háptica, al procesarse en el cerebro son entendidas como un mundo coherente en donde se pueden realizar diferentes acciones interactivas, lo cual hace que el usuario se sienta dentro del mundo virtual.

La inmersión puede lograr que el usuario desarrolle reacciones emocionales al sentirse profundamente envuelto en el sistema virtual.

2.4. Realidad Virtual y sus aplicaciones en la enseñanza

La Realidad Virtual con todas las características que posee, se puede aplicar a la enseñanza como un reforzador y enriquecedor del aprendizaje, dado que podemos crear un software que capture la atención del alumno mostrandole de forma correcta la teoría, en este caso de *Tectónica de Placas*, en una representación en 3D que trate de cumplir con inmersión y presencia, ya que se puede captar mejor la información si vemos en tiempo real su representación y aplicación.

Con este trabajo se pretende apoyar el aprendizaje principalmente de los alumnos de Ciencias de la Tierra, para que puedan visualizar de forma más amena el tema al que se refiere este software.

Capítulo 3

Preparación de Datos

3.1. Datos de las Placas Tectónicas

Para poder lograr la simulación de las placas tectónicas fue indispensable contar con los datos de rotaciones y representación gráfica a lo largo del tiempo de las placas tectónicas, dado que estos son la base de la simulación y la mejor forma que se tuvo para lograr la mayor exactitud posible, los datos con los que se trabajo fueron los siguientes:

- Polígonos de las fronteras de las placas tectónicas de corteza continental.
- Rotaciones y polos de Euler correspondientes a las placas.
- Mapa de alturas del lecho océanico, que se obtuvo del atlas digital de batimetría GEBCO 08.

De los datos de polígonos y rotaciones de placas tectónicas se obtuvieron varios modelos:

- Los primeros datos de polígonos y rotaciones de placas tectónicas fueron obtenidos por cortesía de académicos del Instituto de Geofísica de la Universidad de Texas (Figura 3.1). El conjunto de datos de las placas tectónicas actuales de la corteza continental contiene cerca de 500 polígonos con mucho detalle. Las rotaciones asociadas a dichos polígonos en este conjunto de datos, corresponden a los periodos de hace 750, 230, 165, 100, 65 y 0 (presente) millones de años.

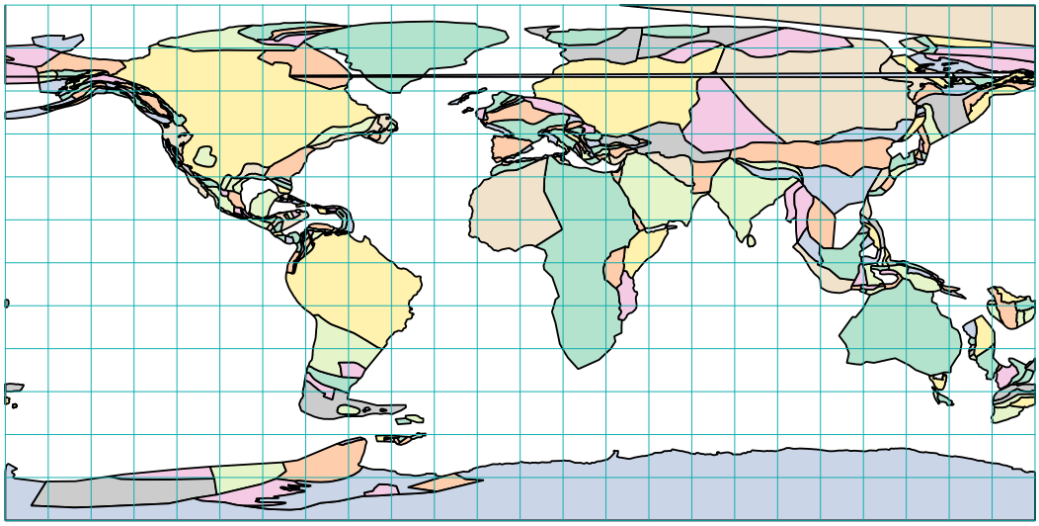

Figura 3.1: Polígonos de placas tectónicas, Universidad de Texas (Sierra et al. [8])

- El conjunto de datos del proyecto Paleomap. Este conjunto de datos cuenta con 216 polígonos con menos detalle que los anteriores. Además, los datos de rotaciones de las placas abarcan desde el presente hasta el Jurásico (hace 300 millones de años) en intervalos de 10 millones de años y se cuenta con proyecciones a intervalos irregulares, al futuro hasta dentro de 250 millones de años (Pangea Próxima) y del Jurásico hasta el Precámbrico (hace 750 millones de años). Algunos de los polígonos de estos datos tenían errores que se corrigieron, y las reconstrucciones anteriores a Pangea no corresponden a modelos de la historia geológica de la Tierra publicados recientemente.
- El conjunto de datos del proyecto Scotese del año 2009 (Figura 3.2).

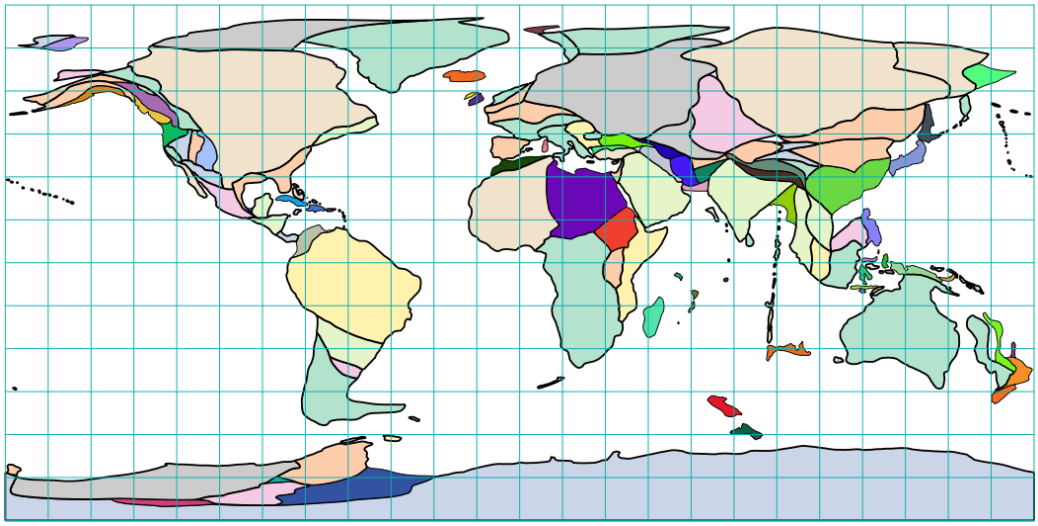

Figura 3.2: Polígonos de placas tectónicas, Scotese 2009 (Sierra et al. [8])

- El conjunto de datos del proyecto Earth Byte (Figura 3.3), con menos detalle que el modelo de Scotese y un menor número de polígonos, 179, pero más actuales y de acceso abierto.

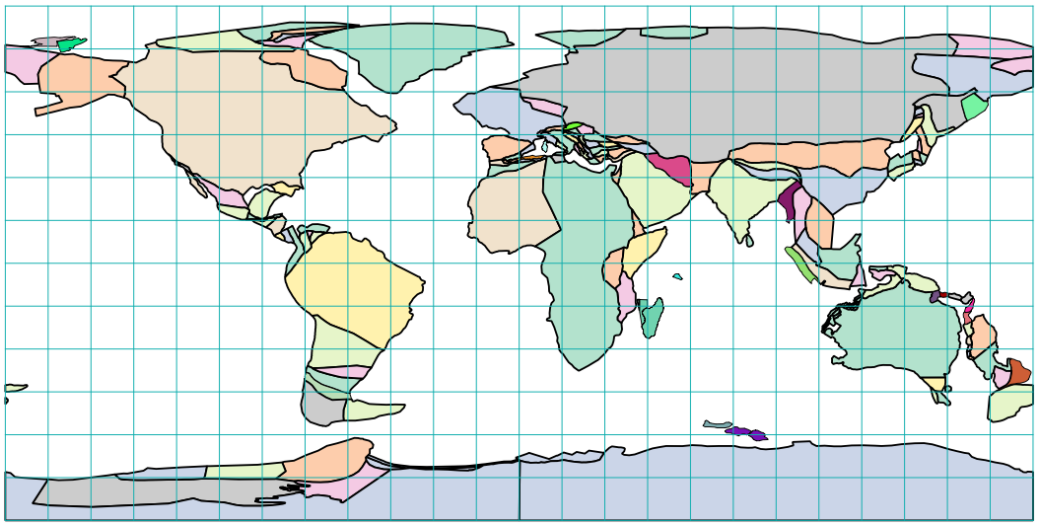

Figura 3.3: Polígonos de placas tectónicas, EarthByte 2009 (Sierra et al. [8])

3.2. Construcción de Mallas 3D a Partir de Polígonos 2D

De los datos que se obtuvieron para la representación de las placas tectónicas los polígonos de las placas fueron convertidos a mallas triangulares, ya que solo teniendo el polígono solo se puede visualizar el borde, lo cual no aporta visualización efectiva de las placas al momento de poner éstas en una representación tridimensional pues no se tendría detalle de la curvatura o se haría de forma erronea.

Para llevar a cabo el cambio de polígonos a mallas triangulares se realizó lo siguiente:

1. Los polígonos en 2D de las placas tectónicas representan placas rígidas sobre una esfera, en proyección cilíndrica en el rectángulo (180W, 90S) - (180E, 90N). En dicha proyección, conforme un objeto se aleja del Ecuador rumbo a los polos, se deforma más, hasta llegar a la anomalía de que un solo punto en el polo se representa a lo largo de toda una línea en los extremos verticales del mapa. En los extremos horizontales ocurre la otra anomalía de que lo que sale por uno entra por el extremo opuesto. Para minimizar la deformación de la placa, primero transportamos el polígono de manera que su centro se encuentre en el Ecuador y a la longitud 0, al centro del mapa, usando una rotación de Euler.

2. Trasladado el polígono al centro del mapa, aplicamos un algoritmo de generación de malla triangular. Con este algoritmo podemos controlar la densidad de triángulos en la malla de manera que su calidad sea suficiente para la escala en que se va a visualizar, lo cual depende de la aplicación que se le vaya a dar al programa. Para una simulación en tiempo real a escala planetaria como en este caso, basta con una densidad moderada.

3. Una vez creada la malla en 2D sobre el mapa, con sus vértices indicando la longitud en la horizontal y la latitud en la vertical, pasamos cada uno de sus vértices a la representación tridimensional, usando la ecuación paramétrica de la esfera.

4. Ya en 3D, transladamos la placa de vuelta a su posición original, usando exactamente la rotación inversa a la usada para colocarla al centro del mapa.

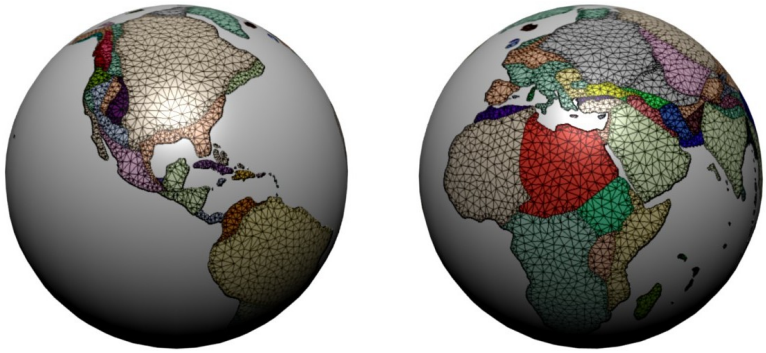

Figura 3.4: Resultado de la triangularización de las placas tectónicas, usando los datos de los polígonos de Scotese. (Sierra et al. [8])

5. Finalmente, el resultado se guarda en disco en formato OBJ, que es un formato de guardado muy simple y con codificación ASCII, además este formato no depende de programas especializados para poder ser leído ni se tiene que pagar por su uso y es soportado por cualquier aplicación de modelado y animación 3D.

3.3. Nomenclatura de las Placas Tectónicas

Cada conjunto de datos sobre las rotaciones de las placas tectónicas identifica a cada placa con un ID, lo cual hace que cada placa tenga un número que la distingue de las demás placas, dicho número es utilizado en los datos de las rotaciones para asignar una latitud, longitud y ángulo a la placa correspondiente.

El uso del ID resulta conveniente dado que no todas las placas tectónicas tienen un nombre bien definido o aceptado por toda la comunidad científica, aunque hubiesen nombres estandarizados, escribir los nombres como forma de identificación de las placas provocaría más problemas que soluciones ya que se tendría que tener en cuenta el uso de mayúsculas y minúsculas asi como el idioma en el cual van a ser escritas. por lo cual el ID resulta la forma mas conveniente de hacerlo.

La numeración de las placas mediante su ID tampoco es un proceso general, por lo que la asignación de cada número dependerá completamente de

cada grupo científico que genere dichos datos. Usualmente se comenzará por Estados Unidos ya que la mayoría de los datos provienen de este país.

Para los propósitos de este trabajo no provoca conflicto alguno el tener diferentes asignaciónes de ID para cada conjunto de datos, ya que los datos leídos se mapean directamente con la placa tridimensional del mismo ID.

Capítulo 4

Desarrollo del Sistema

Una vez que se tienen las herramientas teóricas, es momento de ver en qué forma serán aplicadas al software, para ésto nos servirá el diseño, el cual encaminará la teoría a la aplicación y podremos empezar con el desarrollo.

El objetivo será construir un software que pueda explicar de manera interactiva y clara a los alumnos de Ciencias de la Tierra e Ingenierías afines el funcionamiento de la tectónica de placas.

4.1. Captura de Requerimientos

Para que el software cumpla de la mejor manera posible con su propósito, es recomendable que el usuario al que está enfocado este software de su propia versión de lo que espera de él, a esto se le llama *Relatos del usuario*, los que están descritos en lenguaje natural del usuario y serán la base para empezar a diseñar el software, para obtener dichos relatos se consultó a una doctora en Sismología y Física del interior de la Tierra del instituto de Geofísica, quien da clases a estudiantes de Ciencias de la Tierra, siendo la usuaria idónea para obtener las características más importantes para que el uso de este proyecto fuera de utilidad a los alumnos. Los *Relatos del usuario* para este caso son los siguientes:

- La representación terreste junto con las placas tectónicas, tendrá que poder verse en dos formatos : Esférica y Mollweide.
- Se espera que las placas tectónicas se pueden mover a lo largo del tiempo desde la Pangea hasta la actualidad, reflejando los cambios en

la representación terreste.

- Debe ser posible mostrar las zonas de las fronteras de las placas tectónicas e identificarlas entre convergente y divergente.
- Se pueda activar o desactivar la vista del fondo oceánico donde se mostrarán las dorsales oceánicas.
- El programa permitirá que se pueda seleccionar cualquier placa tectónica resaltandola y obteniendo su información.

Un vez obtenidos los *Relatos del Usuario* se procede con el analisis de requerimientos que serán obtenidos de dichos relatos.

4.1.1. Diagrama de Casos de Uso

Teniendo en cuenta los requerimientos que se tienen que cubrir se crea uno o varios diagramas que organizan las acciones posibles dentro del programa que satisfacen los requerimientos de manera concisa, siendo una guía para estructurar el programa. El número de diagramas depende de que tan grande es el software que se desea crear y la forma en la que las acciones estan relacionadas entre si. Los elementos del diagrama son:

- **Actores:** son la representación de los usuarios, estos pueden ser uno o varios ya que puede que no todos tengan acceso a todas las acciones.
- **Casos de Uso:** son las acciones que se pueden hacer en el software definidas por los *Relatos del Usuario*, dentro del diagrama estos casos están representados dentro elipses.

El diagrama para este software es el siguiente (Figura 4.1):

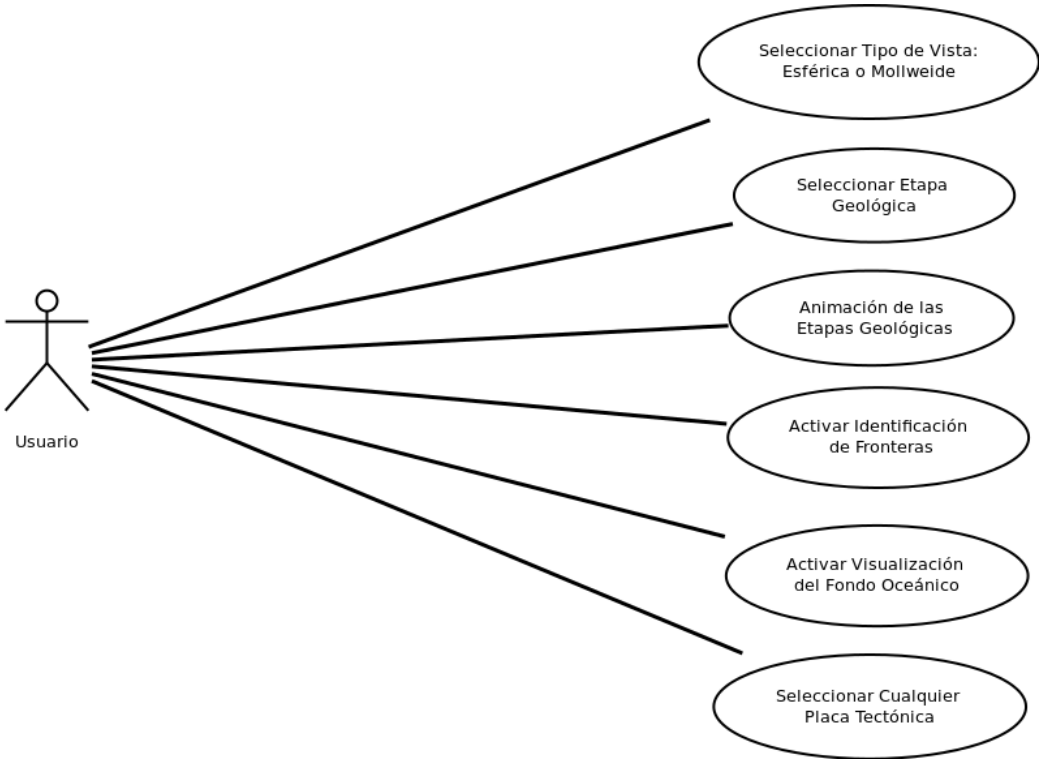

Figura 4.1: Diagrama de Casos de Uso

4.1.2. Análisis de Requerimientos

El *Análisis de Requerimientos* es la parte en la cual se convierten los relatos de usuario con ayuda del *Diagrama de Casos de Uso* a un lenguaje en el cual se introducen conceptos computacionales, estos requerimientos son un guión al momento de desarrollar el software, permitiendo que el diseño del software sea el adecuando para lograr que se cumplan las expectativas que el usuario pidio. Los requerimientos son los siguientes:

- Se requieren dos vistas de las placas tectónicas: Esférica y Mollweide. Estas vistas no afectan la parte lógica del programa por lo cual la forma en la que funcionarán las dos vistas será la misma, además el cambio entre las dos vistas será en tiempo real, puesto que lo único que se modificará será la representación de datos en pantalla.

- Para lograr que las placas se muevan a lo largo del tiempo establecido

se usarán *Cuaterniones*, los datos que se usaran para estas transformaciones serán datos existentes que están divididos cada n millones de años, dependiendo de la exactitud de los datos con la que se cuente, la indexación de los datos cada n millones se hará al momento de ejecutar el programa.

- El movimiento de las placas será posible verlo en una etapa geológica específica, para que el usuario pueda concentrarse en esa etapa, permitiendo una intereacción más específica en las etapas que el usuario considere de mayor interés, por lo cual se requiere de un elemento gráfico que permita realizar esta opción.
- Existirá una animación entre cada uno de los movimientos de las placas a través de las etapas geológicas, permitiendo ver el movimiento de las placas desde diferentes vistas mientras la animación se reproduce.
- Se hará una identificacion del tipo de las fronteras de las placas que podrán ser : Divergente, Convergente y Tranformación; esta información será puesta por encima de las placas para que sea visible al usuario y que pueda activar o desactivar.
- Se podrá visualizar el fondo oceánico en cualquier momento durante el movimiento de las placa tectónicas, usando texturas y la técnica de Bump Mapping[1].
- Cuando el usuario seleccione cualquier placa con el ratón, esta placa se resaltará con un color diferente o un contorno para indicar que ha sido seleccionada y mostrará la información que se tenga de esta placa.

4.2. Diseño

Con toda la información anterior ahora se puede empezar a diseñar el software, el diseño permite identificar las partes o características más importantes del software y la forma en la cual será la interacción entre las diferentes partes. A continuación se detallará la arquitectura del software.

[1]Bump Mapping es una técnica de graficación por computadora en donde se crea la ilusión de que una superficie plana tiene relieve.

4.2.1. Arquitectura de Software

La arquitectura de software es el diseño de más alto nivel. Consiste en definir cuáles serán los componentes que formarán el software. La arquitectura debe favorecer el cumplimiento de los requerimientos funcionales y no funcionales especificados para el producto (Ibargüengoitia and Oktaba [4]).

Una *arquitectura de software* facilita la implementación ya que tenemos el plano de cómo se construirá el programa, tener una arquitectura permite desde un inicio incluir en el software requerimientos para la agregación de nuevos elementos a futuro y la posibilidad de tener un mantenimiento adecuado del software al separar de forma adecuada el código.

Para este software se escogió la arquitectura Modelo Vista Controlador.

4.2.2. Modelo Vista Controlador (MVC)

El Modelo Vista Controlador es un patrón de *arquitectura de software* en el que se separan los componentes del programa en 3 componentes distintos:

- Modelo: consiste de los datos de la aplicación y la lógica del programa. Un modelo notifica a las diferentes vistas que se tenga del modelo y al controlador cuando ha cambiado de estado, esto permite a la vista actualizar la salida y al controlador actualizar el estado de los eventos si fuera necesario. Una implementación pasiva del MVC no manda notificaciones desde el modelo ya sea porque el programa no las necesitas o la plataforma de software sobre la cual se desarrolla no soporta esto.

- Vista: es una representación de la salida de datos, visible para el usuario en pantalla, al separar la vista ésta puede tener varias representaciones o puede estar asociada a varios modelos.

- Controlador: Es el componente encargado de gestionar los eventos que el usuario pueda producir y que estén soportados dentro del programa, mandando las señales correspondientes al Modelo y Vista para mostrar la realización del evento que el usuario haya hecho produciendo la actualización correspondiente al evento que el usuario hizo.

- La principal idea detrás del MVC es la reutilización del código y la separación de código dependiendo del componente al que pertenezca.

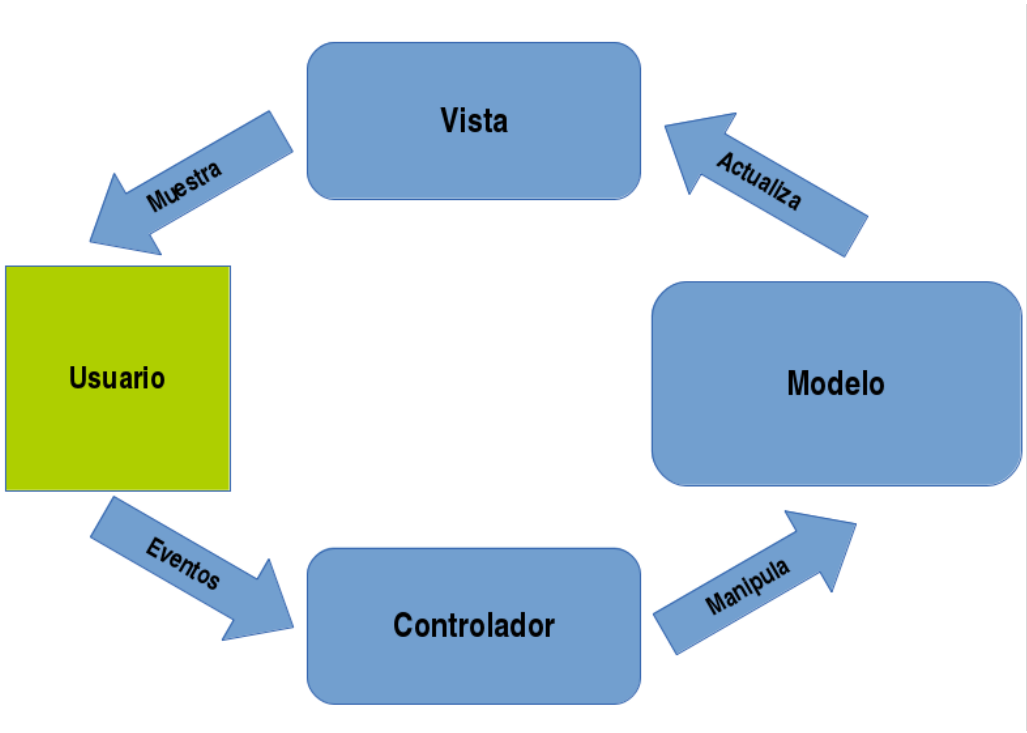

Figura 4.2: Esquema de un Modelo Vista Controlador típico

4.3. Diagrama de Paquetes

Un paquete es un mecanismo general de UML para organizar elementos en grupos. Los paquetes sirven para representar las capas de la arquitectura[4]. Los elementos de cada paquete suelen estar fuertemente ligados entre si, ya que las tareas que realizan son para un fin en común. En cambio el ligado de elementos entre paquetes debe de ser débil para lograr independencia entre los diferentes componentes de la arquitectura y lograr el objetivo de reutilizar el código.

El Diagrama de Paquetes para este programa tomando en cuenta la arquitectura queda de la siguiente manera:

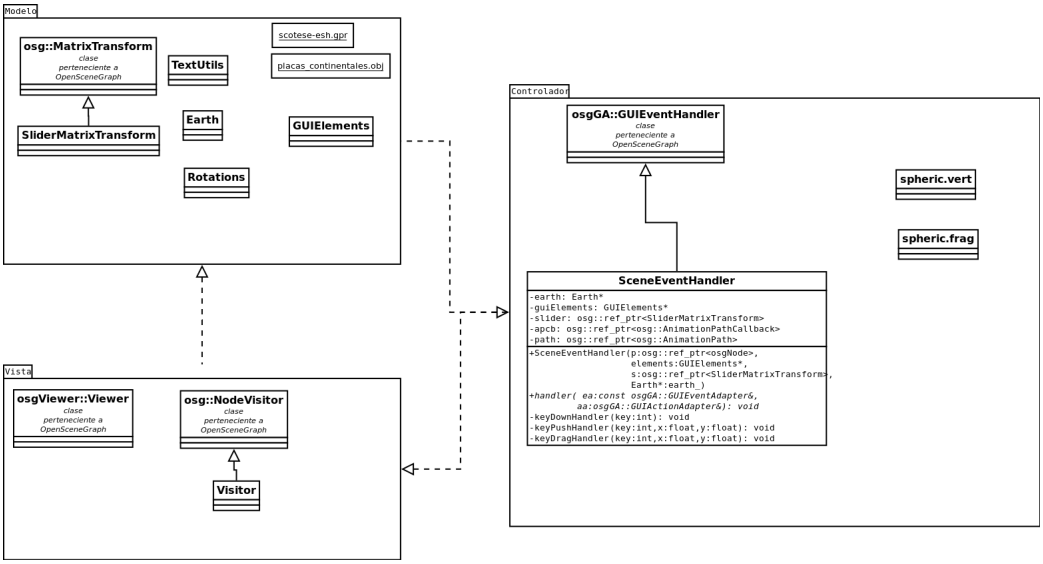

Figura 4.3: Diagrama de Paquetes para el simulador de placas tectónicas

Capítulo 5

Construcción

Una vez que se cuenta con los documentos e información necesaria que nos permitirán crear el software abarcando todos los elementos que se quieren, es momento de transformar esos documentos en software por medio de la escritura del código necesario que nos lleve a eso, para lograr este cometido haremos uso de APIs y Frameworks Gráficos que se explicarán brevemente antes de entrar de lleno a la construcción del software con el uso de los elementos mencionados anteriormente.

5.1. Introducción

Una escena que se desea dibujar como gráficas tridimensionales está compuesta por objetos separados, los elementos geométricos de los objetos son representados por un conjunto de vértices y por primitivas gráficas. Estas primitivas indican el orden de conexión de los vértices necesario para generar una figura.

El hardware especializado en gráficos es capaz de dibujar puntos individuales, segmentos de linea y polígonos rellenos, por lo que cualquier figura o modelo 3D que se desee dibujar esta representado por un conjunto de triángulos, los cuales se forman tomando 3 vértices de la lista de vértices del objeto en el orden que indiquen las primitivas gráficas (Figura 5.1).

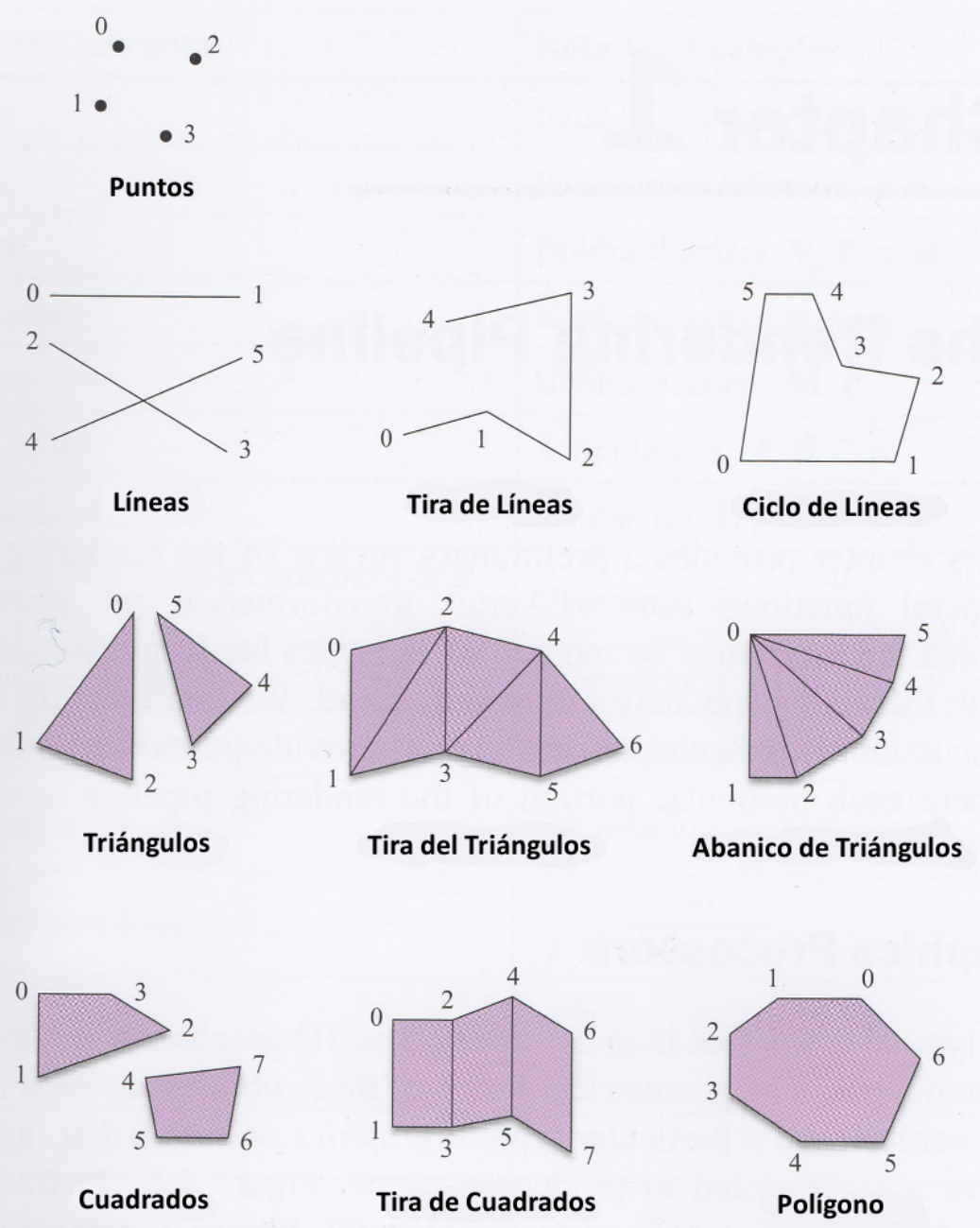

Figura 5.1: Primitivas Gráficas (Lengyel [6]).

Las tarjetas gráficas actuales contienen una unidad de procesamiento grá-

fico (GPU por sus siglas en inglés), que ejecuta instrucciones independientemente del CPU, con lo cual el CPU envía comandos de dibujado al GPU, el cual realiza las acciones requeridas para procesar los comandos mientras el CPU continúa haciendo otras tareas, esto ayuda a que el sistema no se sature con instrucciones de dibujado de elementos geométricos. Cuando la GPU termina de procesar los comandos recibidos, ésta envía la información al CPU para que se despliegue la información en pantalla.

Una aplicación se comunica con el GPU por medio de comandos como se mencionó anteriormente, estos comandos antes de llegar al GPU pasan primero por una biblioteca de dibujado tales como: OpenGL o DirectX, las cuales envían los comandos a un controlador que es el que transforma los comando al lenguaje nativo de la GPU.

Aunque tanto OpenGL como DirectX pueden ser usados en C++, para el desarrollo de este software se usará OpenGL como biblioteca base de dibujado ya que es multiplataforma a diferencia de DirectX que sólo funciona en Windows, por lo cual se omite su uso.

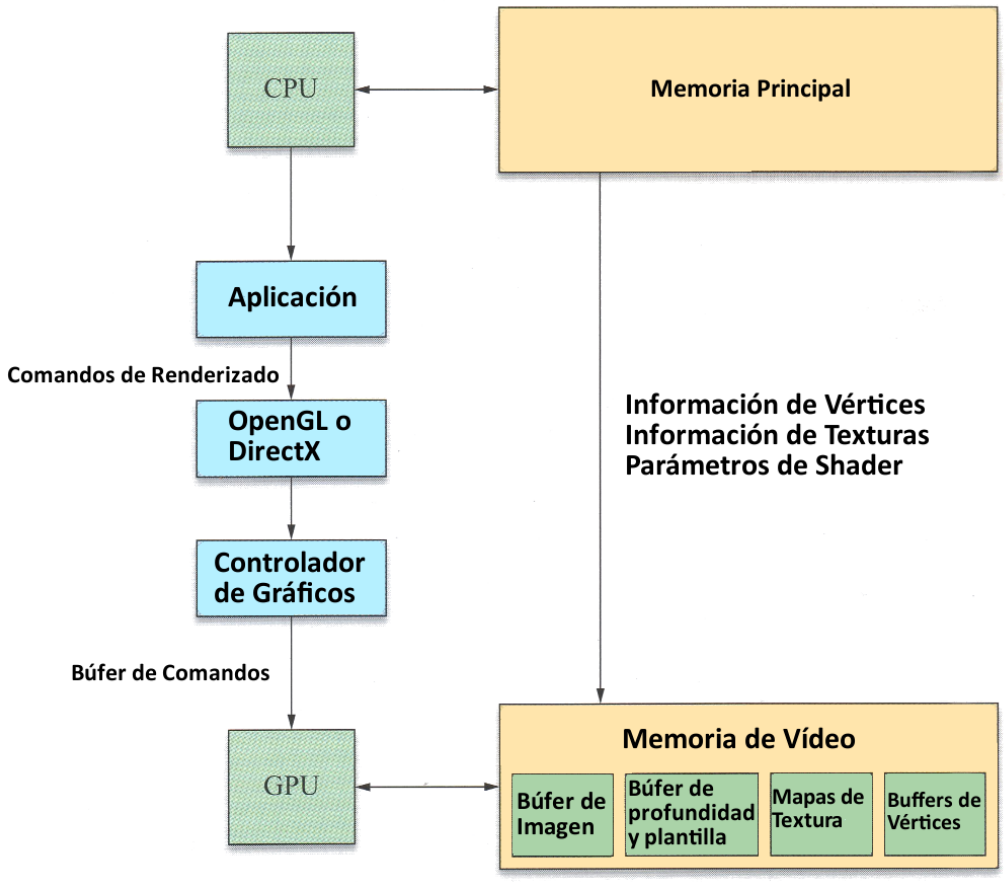

Figura 5.2: Comunicación efectuada entre el CPU y la GPU (Lengyel [6]).

5.1.1. OpenGL

OpenGL es una interface de software que nos permite comunicarnos con las tarjetas gráficas. Para poder lograr esta comunicacion, esta biblioteca de dibujado contiene aproximadamente 700 comandos diferentes que son usados para especificar las operaciones necesarias para producir objetos tridimensionales, lo cual es lo que queremos hacer.

Esta biblioteca esta diseñada para ser independiente del hardware gráfico, lo cual la hace compatible con la mayoría del hardware gráfico disponible. Para que esto sea posible OpenGL no contiene comandos para manejar tareas de ventanas, o para leer entradas del usuario.

OpenGL tampoco contiene comandos de alto nivel para describir modelos tridimensionales complejos como: autos, aviones, mundos de videojuegos, etc. Si se quisiera describir este tipo de modelos más complicados, éstos deben de estar descritos en primitivas geométricas para que pueda ser construido y mostrado.

Como se puede ver esta biblioteca es una pieza fundamental para lograr comunicarnos con el hardware encargado del dibujado tridimensional del programa, sin embargo, aunque nos permita hacer todo lo necesario para el dibujado correcto con gran libertad, todo ésto se hará a un nivel muy bajo de complejidad por lo que todo lo que haga falta corre por nuestra cuenta en cada programa que hagamos. Esto puede llevar a un proceso largo y lento al menos la primera vez que se hagan las cosas, pues podríamos reutilizar código, tomando ventaja de la Orientación a Objetos, pero ésto lleva a otro problema y es que se tendría que planear con mucho cuidado la construccion del código para que sea suficientemente general y pueda ser reutilizado.

5.1.2. OpenSceneGraph

Una API media de renderizado es una API que eleva el nivel de abstracción y elimina las complejidades que se presentan en una API de bajo nivel como OpenGL, el costo de hacer eso es perder flexibilidad sobre cómo se puede hacer el software. Dentro de esta API de nivel más alto estan presentes casi siempre dentro de su diseño la modularidad y la orientación a objetos, aplicados al dibujado de primitivas básicas, materiales etc. Esto permite ahorrar tiempo dentro del desarrollo del software, además de todas las caracteristicas que contiene la API, el diseño es capaz de tomar nuevas funcionalidades como módulos y plugins.

OpenSceneGraph es una API media de renderizado escrita en C++. Es un sistema basado en un renderizado diferido basado en la teoría de los grafos de escena, ésto funciona guardando los comandos de dibujado y los datos en un buffer, para que ésto sea ejecutado en un momento posterior, permitiendo al sistema realizar diferentes optimizaciones a lo que se va a dibujar antes de ser mostrado, aunado a esto se implementa una estrategia de multihilos para mejorar el manejo de escenas complejas Wang and Qian [9].

5.1.2.1. Grafos de Escena

Un grafo de escena es una estructura de datos general que permite definir una relación espacial y lógica de la escena gráfica que se mostrará en pantalla, permitiendo eficiencia en el manejo y renderizado de la información gráfica. Los grafos de escena son típicamente representados como un grafo jerárquico (puede pensarse como un árbol), que contiene una colección de nodos gráficos, incluyendo un nodo principal llamado raíz, un número de nodos grupo, los cuales pueden tener cualquier número de nodos hijo y un conjunto de nodos hoja, estos últimos no tienen nodos hijo y son la capa final dentro del grafo. Un grafo de escena no permite que algún nodo esté conectado con otros generando una cadena cerrada de nodos, o un nodo aislado que no tenga ni hijos ni padres. Cada grupo de nodos puede tener n número de hijos, agrupando esos nodos hijo permite intercambiar información del padre y ser tratados como uno solo. Por defecto una operación realizada en el padre se propagará a todos sus hijos. Cuando un nodo tiene más de un padre se dice que el nodo es "instanciado" permitiendo el comparar la información y el paso de propiedades de cada padre, haciendo al nodo más rico en características, lo cual llega a ser de gran ayuda.

5.1.3. GLSL

GLSL es la abreviación de las siglas en ingles de: OpenGL Shanding Language, el cual es un lenguage de programación basado en la sintaxis del lenguaje C que permite escribir código que será ejecutado directamente en la tarjeta gráfica de la computadora, para hacer ésto, el código es ejecutado en dos etapas:

- Cuando el pipeline del dibujo se encuentra procesando los vértices.
- Cuando toda la escena que se va a dibujar está siendo calculada por pixel.

Este tipo de lenguaje es soportado en todas las tarjetas gráficas que soporten OpenGL.

GLSL permite agregar una mejora visual en la escena sin que sea tan pesado el cálculo de los efectos que se agregan ya que al ser calculados en la tarjeta gráfica el procesador es liberado de estas tareas, además de que el procesador gráfico está optimizado para realizar este tipo de operaciones.

En este programa la funcionalidad de este lenguaje de programación servirá para asignar el color a las placas tectónicas y para crear efectos de relieve sobre la corteza terrestre.

5.2. Grafo General de Escena

Como se mencionó anteriormente, el programa está basado en grafos de escena ya que su utiliza el framework de *OpenSceneGraph*, por tanto lo primero que se necesita es crear el grafo de escena de todo el sistema.

El grafo general tiene un nodo dentro del cual todos los demás nodos están contenidos, este nodo es el que se agrega al objeto osgViewer::Viewer, este objeto se encarga de leer el grafo y mostrar en pantalla todos los elementos que estén dentro de este.

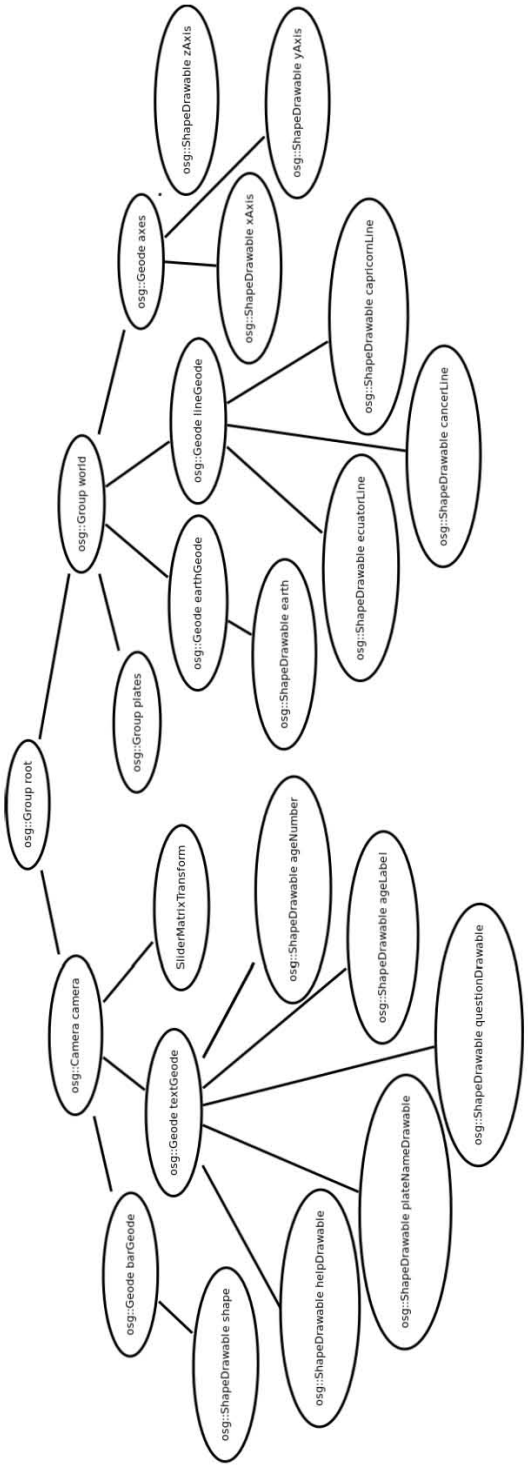

Figura 5.3: Grafo de escena general

Dentro de nuestro diagrama encontramos que tenemos diferentes tipos de nodos y objetos: Group, Geode, ShapeDrawable.

- Los nodos de tipo Group nos permiten agrupar otro tipos de nodos permitiendo que lo que afecte al nodo Group afecte a los hijos dentro de este sin necesidad alguna de otro procedimiento.

- Los objetos de tipo ShapeDrawable son objetos que heredan de la clase virtual Drawable, de esta clase padre se desprenden varias subclases que permiten dibujar modelos tridimensionales, imágenes y texto dentro del pipeline de OpenGL. Los objetos de tipo Drawable al no ser nodos tienen que agregarse a un nodo especial que es Geode.

- Los nodos de tipo Geode (Geometry Node) son nodos que contienen información de geometría dibujable y que funcionan como nodos hoja dentro del árbol, pues a los nodos Geode no se le agregan más nodos sino objetos de tipo Drawable. Los nodos Geode pueden tener cualquier cantidad de objetos Drawable, que serán dibujados cuando OpenSceneGraph lea la información del nodo Geode.

Además en el grafo existen nodos en los cuales se agregan una gran cantidad de nodos, los cuales no son puestos en este diagrama ya que su funcionalidad no es de gran importancia o porque son cargados dimámicamente al momento de ejecutar el programa dependiendo de la información que se tenga, estos nodos son: textGeode y plates.

En el nodo textGeode se tienen nodos para cada raya que indica cierta cantidad de años y el año correspondiente.

Para plates, es el nodo que contiene cada una de las placas tectónicas, por lo cual el número de nodos es dependiente de los elementos con los que se cuente dentro del elemento .obj.

5.3. Creación de la Tierra y Texturas

Como se vio anteriormente lo más importante es tener nuestra estructura de arbol que sera visualizada. La estructura del grafo para llevar a cabo esta tarea es una de las ramas del grafo general se encuentra en la siguiente figura:

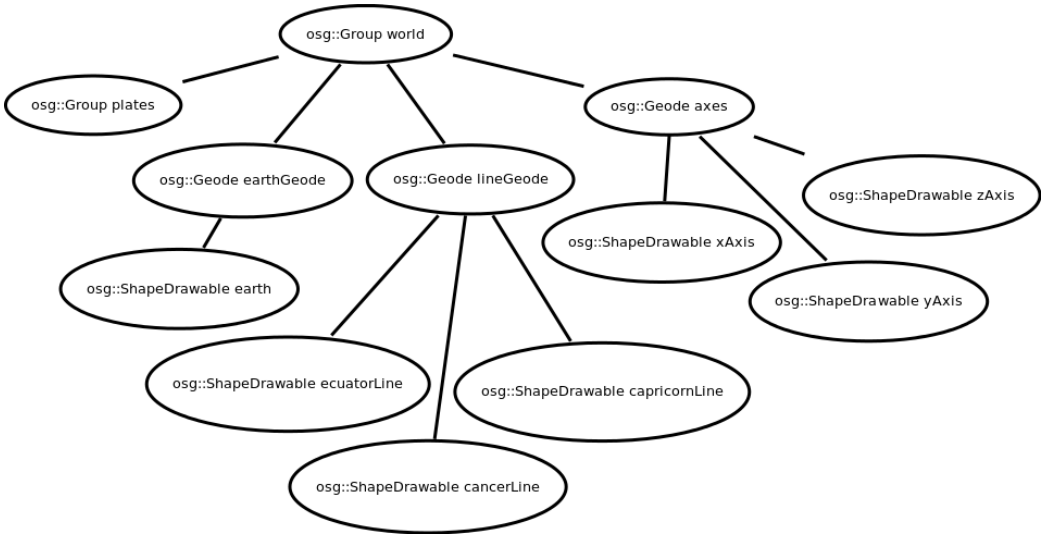

Figura 5.4: Diagrama de Nodos del Globo Terráqueo

Como deseamos modelar el mundo, tendremos nuestro nodo raíz llamado world, al cual le agregaremos 4 nodos:

- Un nodo *osg::Group* llamado plates, en el cual le cargaremos de forma dinámica las placas agregando por cada placa tectónica un nodo de tipo *osg::MatrixTransform*, el cual sirve para rotar las placas, dentro de este nodo se agrega el nodo correspondiente a la información del polígono de la placa. Este procedimiento se lleva a cabo al momento de lanzar el programa por lo cual solo se cuelga el nodo contenedor de las placas y se deja disponible para su uso, a este nodo plates también se le agregan las texturas que podrán tener las placa tectónicas y se carga por default la textura *topobati.jpg*.

- Un nodo *osg::Geode* de nombre earthGeode en el cual como se mecionó tendra como funcion almacenar información dibujable, en este caso será el globo terráqueo. Las texturas correspondientes al fondo oceánico son leidas y la textura *topobatifundo.jpg* se asigna al nodo earthGeode.

- linesGeode que es de tipo *osg::Geode* y que almacenará tres objetos dibujables almacenados cada uno dentro de un *osg::ShapeDrawable*: ecuatorLine, cancerLine, capricornLine. Estos objetos serán tres líneas que estarán en el ecuador, en el trópico de Cancer y en el trópico de Capricornio respectivamente.

- Por último es el *Geode* axesGeode, que también contendrá tres objetos dibujables, en este caso serán los ejes cartesianos x pintado de color azul, y de rojo y z de verde, que sirven como un auxiliar de orientación dentro de la escena. Cabe mencionar que las coordenadas de *OpenSceneGraph* están invertidas en cuanto a eje Y y Z respecto a un espacio cartesiano común.

Para poder utilizar una textura o asignar atributos a un nodo es necesario usar la clase osg::StateSet. OpenGL usa una máquina de estados para llevar un registro de los estados de las cosas que se dibujan; los estados de dibujado son colecciones de atributos como: luces de la escena, materiales, texturas y modos de estado los cuales pueden ser intercambiados entre encendido y apagado mediante las funciones de *OpenGL glEnable()* y *glDisable()*. Cuando un estado de dibujado es puesto, éste se mantiene en ese estado hasta que se cambie de nuevo.

Una instancia de un *osg::StateSet* contiene un subconjunto de diferentes estados de OpenGL, estos estados pueden ser aplicados a un objeto de tipo *osg::Node* u *osg::Drawable*, usando el método setStateSet(). Una forma más segura de realizar la asignación del *osg::StateSet* es con el método: *getOrCreateStateSet*(), el cual regresará un *osg::StateSet* previamente asignado o en su defecto si no existe un *osg::StateSet* previo se crea uno y se asigna al objeto que lo regresa, por lo cual el *osg::StateSet* siempre será válido.

Una textura es un archivo de datos que contiene la información sobre la imagen que despliega dicho archivo, como la proporcion de los colores RGBA por pixel si es una imágen con transparencia o RGB si no tiene, esta información en algunos formatos de imagen puede ser legible pues se encuentra en formato ASCII, pero para los formatos más avanzados y comúnmente usados como .jpg y .png dicha información no es legible pues se encuentra comprimida.

Cuando el programa carga una textura, realmente lo que carga es un arreglo de datos del tamaño de los pixeles que tenga la imagen, ya que al dibujarse los elementos gráficos, estos se dibujan pixel por pixel.

La forma en la que una textura es "pegada" a una geometría es mapeando los datos de la textura con los vértices de la geometria, una forma sencilla de imaginarse esto sería pensar que la textura es un papel de colores que se pone debajo de algunas figuras recortadas, entonces recortamos el papel del color por el contorno de la figura y después la pegamos encima de la figura, de esta forma quedará cubierta con el papel. De una manera similar la forma

de "recortar" el contorno de la geometría es mapeando los vértices contra la posición equivalente dentro de la textura, el tamaño de la textura es unitario ya que sus coordenadas estan entre 0 y 1 no importando el número de pixeles que tenga la imágen.

El cargado de texturas se hace en dos pasos:

- Lectura de la textura: La textura será leída por un método que ya incluye el framework que usamos y que nos regresará un objeto de tipo Image que contiene utilerías comúnmente relacionadas con el manejo y manipulación de imágenes. Con esto se cumple el primer paso que es tener los datos de la textura en el programa.

```
osg::ref_ptr<osg::Image> oceanImg;
osg::ref_ptr<osg::Image> plateImg;
osg::ref_ptr<osg::Image> oceanImg2;
osg::ref_ptr<osg::Image> plateImg2;
oceanImg2 = osgDB::readImageFile("images/edad-termal-2k.png");
oceanImg = osgDB::readImageFile("images/topobatifondo.jpg");
plateImg2 = osgDB::readImageFile("images/placaspastel.png");
plateImg = osgDB::readImageFile("images/topobati.jpg");
```

- Activación de la textura: Ya que se tiene cargada la textura se tienen que asociar a algún nodo que esté dentro de la escena para que tenga efecto sobre las geometrías que esta rama tenga. La forma en la que se puede asociar una imagen con un nodo y con otra variedad de elementos es por medio de un conjunto de estados(StateSet) con el que cada nodo cuenta, en este StateSet se le pasa la imagen cargada y se activa el atributo de la textura para ese nodo y a la textura se le especifica su comportamiento. Cabe mencionar que a cada textura se le da un índice único dentro de todo el programa.

```
oceanTexture = new osg::Texture2D();
texturePlate = new osg::Texture2D();
oceanStateSet = new osg::StateSet();
plateStateSet = new osg::StateSet();
oceanTexEnv = new osg::TexEnv();
plateTexEnv = new osg::TexEnv();
oceanTexture->setDataVariance(osg::Object::DYNAMIC);
oceanTexture->setImage(oceanImg.get());
texturePlate->setDataVariance(osg::Object::DYNAMIC);
```

```
texturePlate->setImage(plateImg.get());
oceanTexEnv->setMode(osg::TexEnv::DECAL); oceanStateSet->
    setTextureAttributeAndModes(0,oceanTexture,osg::
    StateAttribute::ON); oceanStateSet->setTextureAttribute
    (0,oceanTexEnv.release());
plateTexEnv->setMode(osg::TexEnv::DECAL); plateStateSet->
    setTextureAttributeAndModes(1,texturePlate,osg::
    StateAttribute::ON); plateStateSet->setTextureAttribute
    (1,plateTexEnv.release());
earthGeode->setStateSet(oceanStateSet.release());
plates->setStateSet(plateStateSet.release());
```

A continuación se muestra como se ve el programa con esta parte completa, aunque las texturas de las placas ya se han cargado, éstas no son mostradas porque falta cargar las placas tectónicas y asociar a cada nodo de placa con la textura.

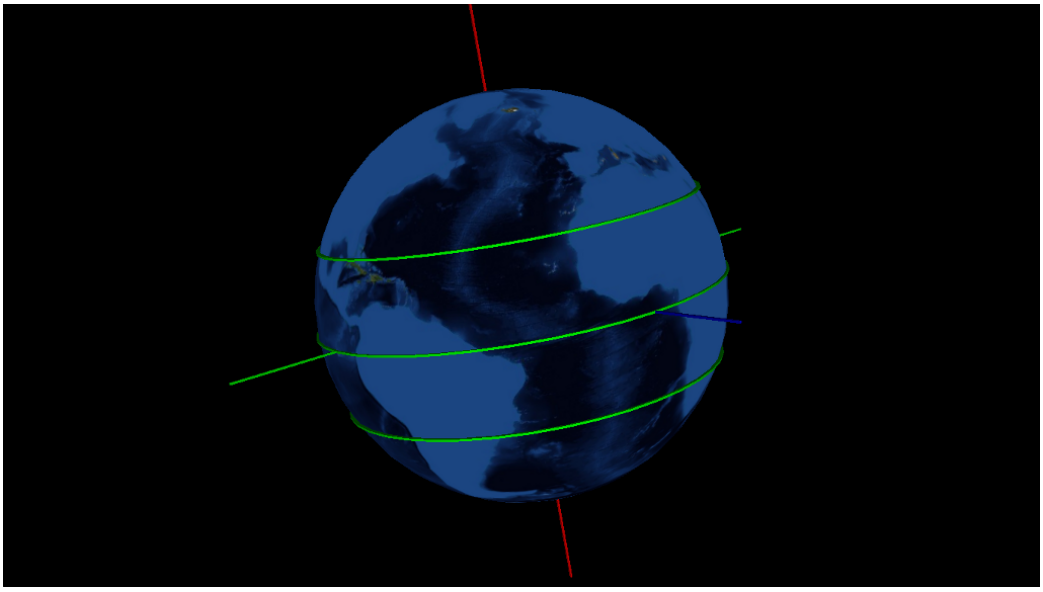

Figura 5.5: Imagen de la Tierra con la textura del fondo oceánico

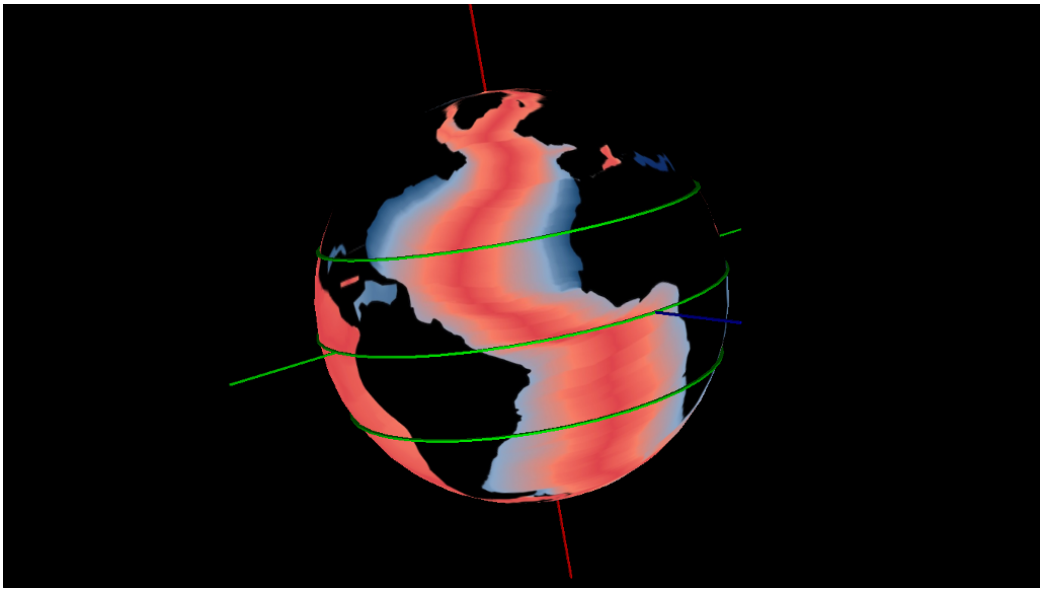

Figura 5.6: Imagen de la Tierra con la textura térmica del océano

5.4. Barra de Edades y Eventos de Usuario

La barra de edades es una parte fundamental dentro de la parte visual en cuanto al manejo de la información dentro del programa, ya que es la manera en la que dentro de la pantalla se refleja y actualiza la edad en la que se encuentran las placas tectónicas. La barra es la principal forma en la que estará interactuando el usuario con el software al momento de usarlo, por lo cual se le debe de poner especial énfasis para que el usuario la tenga presente en todo momento.

Las rotaciones de las placas tectónicas están directamente ligadas con la barra de edades ya que cada cambio en la barra o alguno de sus componentes por la interacción del usuario, las rotaciones se actualizan, reflejando los cambios correspondientes a la edad que se ha elegido para mostrar en pantalla, por este motivo es que estas dos partes se hacen en la misma sección.

El grafo de escena para ésta sección es el siguiente:

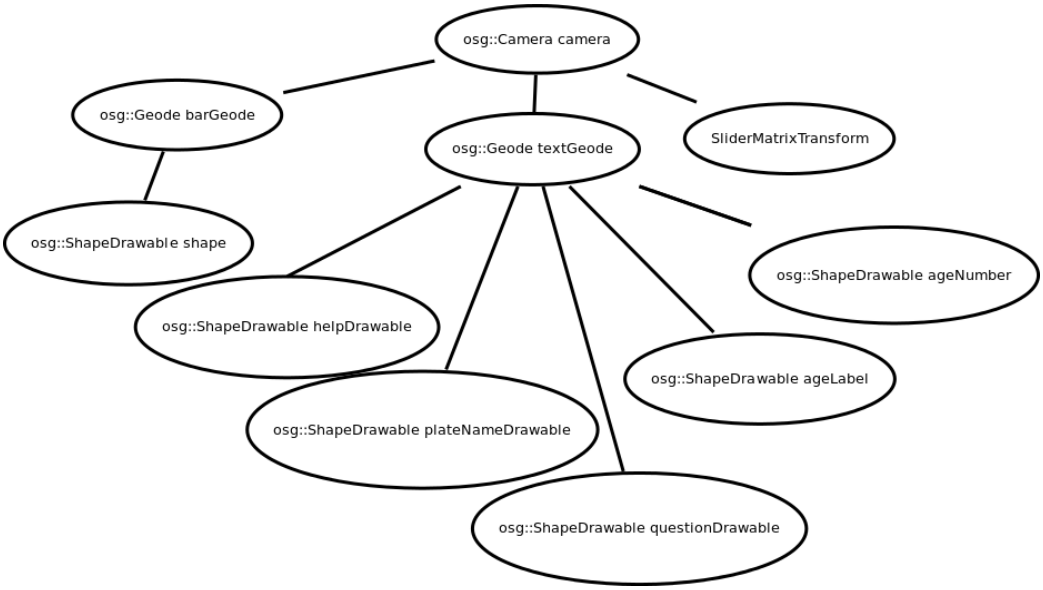

Figura 5.7: Diagrama de los nodos de los elementos de interface gráfica

La inclusión de los elementos gráficos, se hace de una manera diferente a como se agregan los demás elementos ya que estos elementos estarán por encima de todos los elementos que se mostrarán en pantalla, para lograr ésto los nodos correspondientes serán agregados a un nodo de tipo osg::Camera, al cual se le asignan las siguientes propiedades:

- *setName()* Se le da un nombre al nodo para identificarlo.
- *setReferenceFrame()* Se le dice que su posición será absoluta, esto quiere decir que no se moverá ni se verá afectada por las rotaciones y traslaciones que se hagan en la escena.
- *setClearMask()* Se limpia el buffer de profundidad ya que la cámara será dibujada al final de toda la escena, para permitir posicionar los elementos correctamente y pasen la prueba de profundidad permitiendo tener los elementos al frente de la pantalla.
- *setRenderOrder()* Se indica que los elementos que sean hijos de Camera serán dibujados después de que todos los demás objetos hayan sido dibujados.

- *setAllowEventFocus()* Indicamos que la cámara no reciba ninguna interacción del ratón o del teclado para que no interfiera con la interacción de los controles que el usuario usará.

- *setProjectionMatrix()* Con este método decimos que proyección se utilizará para dibujar los elementos de la cámara, en este caso utilizaremos una proyección ortogonal que dibujará un cuadro de 1280 pixeles de resolución a lo ancho y 1024 pixeles de resolución de alto, en esta resolución se mostrarán todos los elementos asignados a este nodo de tipo *osg::Camera.*

- El estado del atributo de iluminación se apaga para que no exista ningun conflicto cuando se esté manipulando la escena.

- Se activa el atributo blend, para activar transparencias en los elementos.

- Se indica que el estado de la cámara es transparente para que los elementos transparentes sean dibujados despues de los opacos.

```
camera->setName("CameraNode");
camera->setReferenceFrame(osg::Transform::ABSOLUTE_RF );
camera->setClearMask(GL_DEPTH_BUFFER_BIT);
camera->setRenderOrder(osg::Camera::POST_RENDER);
camera->setAllowEventFocus(false);
camera->setProjectionMatrix(osg::Matrix::ortho2D(0,1280, 0, 1024));
camera->getOrCreateStateSet()->setMode(GL_LIGHTING, osg::StateAttribute::
    OFF ); camera->getOrCreateStateSet()->setMode(GL_BLEND, osg::
    StateAttribute::ON); camera->getOrCreateStateSet()->setRenderingHint
    (osg::StateSet::TRANSPARENT_BIN);
```

La barra de edades se compone de dos elementos, la barra en donde se encuentran todos los intervalos de millones de años y un pequeño cuadro rojo que es el que el usuario puede mover para seleccionar la edad que desee.

La barra en donde se encuentran las edades consiste solamente en un rectángulo de color azul sobre el cual se ponen todos los demás elementos.

El cuadro selector que indica la edad que se muestra en la pantalla corresponde a una clase que al momento de moverse actualiza la visualización de la posición de las placas así como el texto que indica los millones en que nos encontramos.

La clase del cuadro selector se llama *SliderMatrixTransform* y hereda de *osg::MatrixTransform,* ésto permite que el selector se mueva por medio de matrices de traslación. Como esta clase hereda de una clase de *OpenSceneGraph* se tiene que implementar el método *traverse()* el cual es llamado al

momento de dibujar el grafo por lo que se aprovecha esta implementación para poner el código que actualiza el texto de la edad seleccionada y la posición de las placas tectónicas para la edad, haciendo esto se simplifica la forma en la que se actualizan estos elementos ya que solo se tiene que trasladar el selector y este calcula la edad correspondiente y actualiza la edad de la clase *Visitor* que se explica posteriormente.

Para la creación del texto se tiene que crear un objeto de tipo osg::Text que contendrá: la tipografía, el tamaño y la posición del texto dentro de la pantalla. El objeto de tipo Text además se agrega a un osg::Geode para que se pueda dibujar como se mencionó anteriormente. Para evitar el problema de que no se cuenta con la tipografía especificada en el sistema, se tiene una carpeta llamada fonts en donde se pueden poner las tipografías que se quieran usar y desde ahí obtenerlas.

La creación y preparación del texto se hacen dentro de un método el cual regresa un osg::Text con todo lo necesario para ser dibujado:

```
osgText::Text* GUIElements::createText(const osg::Vec3& position, const std
    ::string& content, float size ){
  osg::ref_ptr<osgText::Text> text = new osgText::Text();
  osg::ref_ptr<osgText::Font> font = osgText::readFontFile("fonts/arial.ttf"
      );
  text->setFont(font.get());
  text->setCharacterSize(size);
  text->setAxisAlignment(osgText::TextBase::XY_PLANE);
  text->setPosition(position);
  text->setText(content);
  return text.release();
}
```

Ya que la aplicación tiene como destino el uso para estudiantes de México y en general usuarios de habla hispana, se tienen que escribir los textos con acentos en las palabras que lo requieran y ñ en caso de que exista en alguna pregunta; Esto representa un problema ya que el formato que se usa comunmente es ASCII, por lo cual tenemos que usar un tipo especial de char* en C++ el cual es wchar_t* y que nos permite guardar los caractéres especiales, por lo cual se crea otro método que use este tipo de char*

```
osgText::Text* GUIElements::createTextW(const osg::Vec3& position, wchar_t*
    content, float size ){
  osg::ref_ptr<osgText::Text> text = new osgText::Text();
  osg::ref_ptr<osgText::Font> font = osgText::readFontFile("fonts/arial.ttf"
      );
  text->setFont(font.get());
  text->setCharacterSize(size);
  text->setAxisAlignment(osgText::TextBase::XY_PLANE);
  text->setPosition(position);
  text->setText(content);
```

```
  return text.release();
}
```

Estos métodos se utilizaran para poner el texto a la barra de edades dividiendola en intervalos de cada 20 millones de años (Figura 5.8).

```
void GUIElements::drawText(){
  int i;
  textGeode->addDrawable(createTextW(osg::Vec3(520.0f,1000.0f,0.0f),(wchar_t
      *)L"Simulador de Evolución Tectónica", 20.0f));
  for(i =0 ; i <= 32 ; i= i+2){
    textGeode->addDrawable(createText(osg::Vec3(37.0f+((float)i*(1200.0f/
        SECTIONS)),947.0f,0.0f),"|",40.0f));
    textGeode->addDrawable( createText(osg::Vec3(37.0f+ ((float) i* (1200.0f
        /SECTIONS)), 920.0f,0.0f),"|", 40.0f));
    std::stringstream ss;
    ss << 240-i*10;
    if(i >= 10)
      textGeode->addDrawable( createText(osg::Vec3(25.0f+((float) i* (1200.0
          f/SECTIONS)), 890.0f,0.0f),ss.str(), 25.0f));
    else
      textGeode->addDrawable( createText(osg::Vec3(32.0f+((float) i* (1200.0
          f/SECTIONS)), 890.0f,0.0f),ss.str(), 25.0f));
  }
}
```

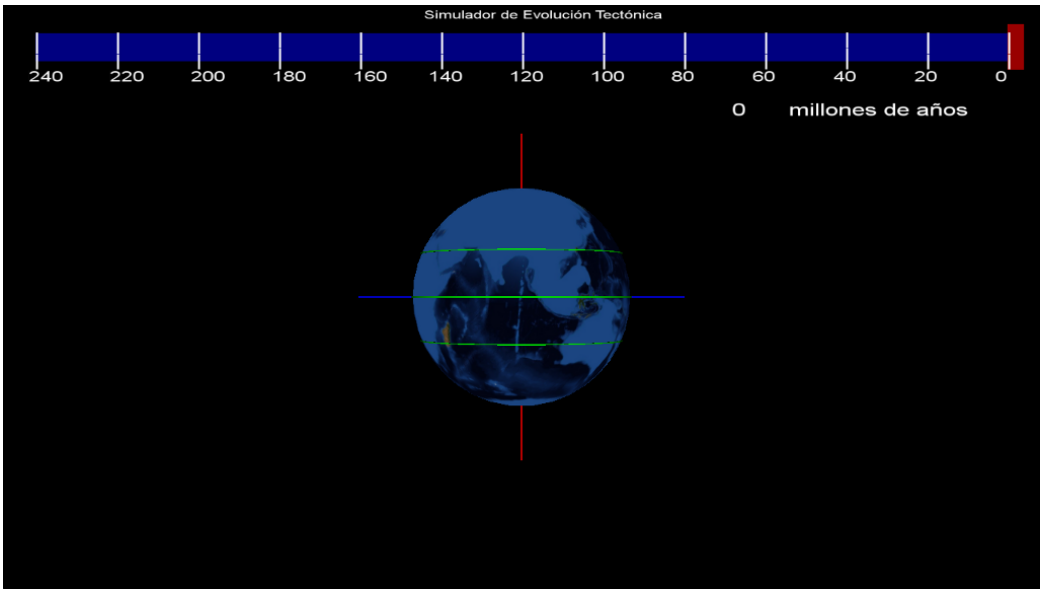

Figura 5.8: Elementos de la interface gráfica agregados al programa

Las placas tectónicas se encuentran dentro del archivo *placas_ continentales.obj*. Este archivo tiene la información de todos los vértices de cada una de las pla-

cas, y estos vértices se encuentran agrupados por placa tectónica. Para poder cargar este archivo y ver los objetos tridimensionales dentro de el programa, se necesita leer el archivo con la información de los vértices en el formato correspondiente a .obj, agregar cada vértice a un *osg::Node* en el caso del framework que se está utilizando, se cuenta con un método dentro *OpenSceneGraph* que hace esto por nosotros dentro del *namespace osgDB* que se llama *readNodeFile()*, el cual regresa un *osg::Node* con la información de todo el archivo que le pasamos, o un osg::Group agrupando los vértices por placa tectónica en nodos hijo. Este último caso es el que se tiene para el archivo .obj. Una vez que se tienen los datos cargados, se tiene que definir en qué forma van a ser dibujados y que respondan a las rotaciones de cada edad. La forma en la que se van a cargar los datos va a ser utilizando un *patrón visitante.*

Un *patrón visitante* se refiere a que un usuario puede realizar alguna acción sobre los elementos de un grafo sin modificarlos, OpenSceneGraph tiene la clase osg::NodeVisitor para poder implementar este patrón. Para poder aplicar este patrón dentro del código es necesario heredar de osg::NodeVisitor, lo cual permitira recorrer un grafo, visitando cada uno de sus nodos y aplicarle las operaciones que necesitaremos para acomodar las placas, la función que se necesita implementar en el hijo que es donde se pueden decir las operaciones que se realizan en los nodos, la función se llama: *osg::NodeVisitor::apply(osg::Node &n).*

La clase que es un hijo de osg::NodeVisitor en el programa es Visitor.hpp, y dentro de la función *apply()* se manipulan los elementos de diferentes maneras:

- Cuando se cargan por primera vez los datos.
- Para actualizar la posición y selección de las placas cuando el usuario interactúa con el programa.

Cuando se cargan por primera vez los datos lo que se hace es primero guardar el nombre que tiene la placa en el .obj dentro de un *std::vector*, para que sea el nombre por default que se mostrará cuando se seleccionen las placas. Se crea otro objeto de tipo *osg::MatrixTransform()* el cual servirá para almacenar la representación del cuaternión, que contiene a su vez la rotación de la placa con la cual se definirá su posición y que además de agregarlo al grafo de escena se agrega también a un *std::map* para su posterior manipulación. Para obtener la rotación dentro de un cuaternión se utiliza la función *void*

getRotation(osg::Quat &q,int code)* que esta dentro de la clase *Rotations*, aunque antes de mandar llamar a esta función se deben de cargar los datos.

La clase *Rotations* es la encargada de obtener los datos de las rotaciones de un archivo binario con extensión .gpr (Global Plate Rotations), el cual mantiene el mismo formato de los datos en texto, pero comprimidos para que los datos ocupen menos espacio ya que el tamaño se puede volver relevante al tratar con muchas placas tectónicas dependiendo de los datos o la información de las rotaciónes cada miles de años y no millones. Los datos del archivo binario se cargan mediante la función *bool readAges(char* filename)*, esta función es llamada dentro del constructor del objeto *Visitor* y la función lee el archivo binario en donde se obtiene la edad de las rotaciones que se leerán, después el código al que pertenece la rotación y finalmente el ángulo, la latitud y longitud de la rotación correspondiente, la cual se transforma a un *osg::Quat*; con esto la información se indexa dentro de un std::map<int , std::map<int , osg::Quat*>>, la idea de guardarlo así es porque la primera llave es la edad, y ésta puede contener múltiples placas tectónicas con sus respectivas rotaciones que corresponde al std::map anidado, además tener almacenados los datos por llaves permite una localización más rápida, lo cual no sucedería con una lista o un *std::vector* ya que los números de las placas no son contínuos ni tienen un orden definido.

En la función *void getRotation(osg::Quat* &q,int code)*, el primer parámetro corresponde a que a esta función se le pasa la dirección del cuaternión en donde nos va a regresar la rotación correspondiente a la placa que tenga el mismo número que el segudo parámetro que se le pasa. Para saber la edad de la cuál se tiene que dar la información, existe una variable global llamada age, de tipo entero el cual se pone en la clase con la función *setAge(int age_)*; con el objeto cuaternión, la edad y el número de placa se buscan los datos primero preguntado si la edad que se desea existe dentro del std::map de edades, si existen los datos para esa edad se busca el número de la placa ya que no todas las placas tienen rotaciones en todas las edades, si el número de la placa se encuentra entonces se pone la información dentro del *cuaternión* que se pasó como parámetro. Si la edad que se necesita no existe entonces se procede a hacer una interpolación esférica que se explicó anteriormente en la sección de cuaterniones. Lo primero que se hace es obtener dos edades que existan, una mayor y una menor a la que deseamos, dentro de esas edades se busca la información que corresponde a la rotación de la placa, y con esta información se crean los cuaterniones que serán $q_1 q_2$, una vez que se tienen los dos cuaterniones necesarios se aplica la ecuación de la interpolación lineal

esférica, en caso de que no se cuente con datos para generar alguno de los cuaterniones, se regresa un cuaternión identidad. El cuaternión resultante es la identidad, una rotación existente, o el resultado de la interpolación esférica.

```
void Rotations::getRotation(osg::Quat* &q,int code){
  bool valid = false, low = false, up = false;
  int age1 = 0, age2 = 0;
  double t,dot, theta, phi;
  osg::Quat* q2;
  osg::Vec3 tmpV3;
  mapPlatesID::iterator codeIter;
  mapPlatesID::iterator codeIter1;
  mapPlatesID::iterator codeIter2;
  std::map<int , mapPlatesID>::iterator ageIter;
  std::map<int , mapPlatesID>::iterator ageIterLow;
  std::map<int , mapPlatesID>::iterator ageIterUp;
  ageIter = mapAges.find(age);
  if(ageIter != mapAges.end()){
    codeIter = ageIter->second.find(code);
    if(codeIter != ageIter->second.end()){
      q = codeIter->second;
      valid = true;
      q->getRotate(phi,tmpV3);
    }
  }else{
    age1 = age;
    while(mapAges.find(age1) == mapAges.end() && age1 >= 0){
      age1--;
    }
    ageIterLow = mapAges.find(age1);
    printf("CODE:%i, AGE: %i AGE1: %i \n",code,age,age1); fflush(
       stdout);
    if(ageIterLow != mapAges.end()) low = true;
    age2 = (*mapAges.upper_bound(age)).first;
    printf("AGE2: %i\n", age2); fflush(stdout);
    ageIterUp = mapAges.find(age2);
    if(ageIterUp != mapAges.end()) up = true;
    if(low && up){
      codeIter1 = ageIterLow->second.find(code);
      codeIter2 = ageIterUp->second.find(code);
      if(codeIter1 != ageIterLow->second.end() && codeIter2 !=
         ageIterUp->second.end() && age1 != age2){
        codeIter1->second->getRotate(phi,tmpV3);
        q = new osg::Quat(phi, tmpV3);
        q->getRotate(phi,tmpV3);
```

```
        codeIter2->second->getRotate(phi,tmpV3);
        q2 = new osg::Quat(phi, tmpV3);
        q2->getRotate(phi,tmpV3);
        t = (double)(age-age1)/(age2-age1);
        dot = q->w()*q2->w() + q->x()*q2->x() + q->y()*q2->y() +
            q->z()*q2->z();
        if(dot < 0.0){
          (*q) *= -1.0;
          dot = q->w()*q2->w() + q->x()*q2->x() + q->y()*q2->y()
              + q->z()*q2->z();
        }
        if(dot != 1.0){
          theta = sqrt(1.0-pow(dot,2.0));
          (*q) *=( (theta*(1.0-t))/theta);
          (*q2) *= ((theta*t)/theta);
          (*q) += (*q2);
          valid = true;
        }
      }
    }
  }
  if(!valid){
     q = new osg::Quat(0.0,osg::Vec3(1.0,1.0,1.0));
  }
}
```

El cuaternión se guarda en su forma matricial dentro de un std::map para accederlo de forma rápida a la hora de responder a los eventos del usuario, a esta matriz también se le cuelga como un hijo el osg::Node correspondiente a la placa, esto para que la rotación afecte a la placa que corresponda, además el StateSet del nodo de la placa se modifica cargándole un shader que, tendrá la función de modificar el color de la placa seleccionada. Finalmente la matriz se agrega al nodo raíz el cual es el *osg::Group* que anteriormente se mencionó serviría para contener las placas tectónicas y que se encuentra dentro de la clase *Earth*.

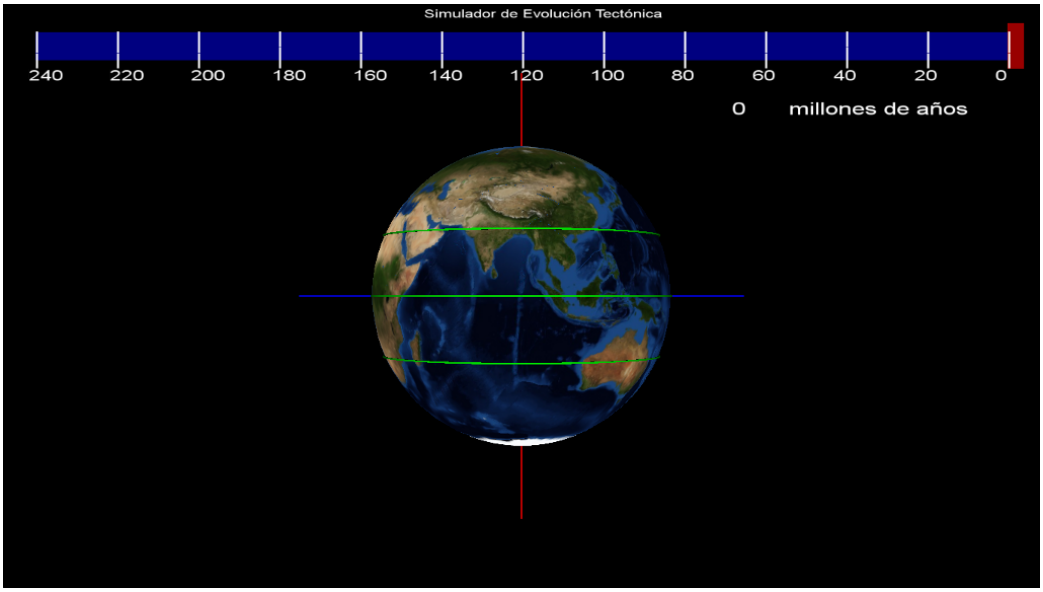

Figura 5.9: Placas tectónicas cargadas y agregadas al grafo de escena

Para que la barra de edades responda a las acciones del usuario, el visor de OpenSceneGraph que es el encargado de visualizar todo, cuenta con una lista de eventos de tipo *osgGA::GUIEventHandler*, que se agregan al *osg::Viewer*. Para poder captar los eventos que se necesitan se crea la clase *SceneEventHandler* que hereda de *osgGA::GUIEventHandler* y donde se implementa la función virtual *virtual bool handle(const osgGA::GUIEventAdapter& ea, osgGA::GUIActionAdapter& aa)* que es en donde se pondrá el manejo de la barra de edades que se explicará a continuación.

El manejador de eventos siempre está monitoreando las teclas o movimientos y clicks del ratón, por lo cual en el *handle()* se va a hacer un switch para saber primero que tipo de evento pasó, si se presionó una tecla ocurre un evento *osgGA::GUIEventAdapter::KEYDOWN*, si se presionó un botón del ratón *osgGA::GUIEventAdapter::PUSH*, cuando se deja de presionar se produce el evento *osgGA::GUIEventAdapter::RELEASE*, al mover el ratón el evento es de tipo *osgGA::GUIEventAdapter::MOVE* o si se dio presionó algún botón y se arrastró el ratón.

Para el caso de las teclas se pregunta por el evento *osgGA::GUIEventAdapter::KEYDOWN* y se hace otro switch para saber cuál es la tecla que se ha presionado y efectuar las operaciones necesarias si es el caso de que sea alguna tecla que nos importa. Las teclas que usaremos son las siguientes:

- *osgGA::GUIEventAdapter::KEY_H.* Cuando se presiona la tecla H se muestra la ayuda, que consiste en un recuadro de color gris sobre el que se muestra el texto en el cual se indica que teclas realizan alguna acción y una descripción breve de dicha acción. Este recuadro se construye dentro de la clase GUIElements, y es llamada desde el manejador

- *osgGA::GUIEventAdapter::KEY_Q.* La tecla Q permite cambiar entre las preguntas que se muestran, cuando se inicia el programa no se muestra ninguna pregunta, hasta que esta tecla es presionada por primera vez y se irá avanzando en las preguntas con cada tecleo.

- *osgGA::GUIEventAdapter::KEY_A.* Inicia una animación del movimiento de las placas tectónicas

- *osgGA::GUIEventAdapter::KEY_O.* Intercambia las texturas del océano del globo terráqueo. Cuando el programa inicia la textura que se muestra es la del fondo oceánico, al presionar esta tecla cambiará por la textura termal de los océanos. Si la tecla se vuelve a presionar se mostrará nuevamente el fondo oceánico.

- osgGA::GUIEventAdapter::KEY_Tab. Intercambia entre las dos texturas disponibles: la textura de la superficie de los continentes y la textura en donde se encuentran coloreadas las placas tectónicas de colores diferentes para que sean visibles los límites fácilmente. Al iniciar el programa la textura que se muestra es la superficie de las placas tectónicas.

- osgGA::GUIEventAdapter::KEY_P. Pausa y reanuda la animación dependiendo en que estado se encuentre la animación.

- osgGA::GUIEventAdapter::KEY_Delete. Capta el evento relacionado con la tecla *supr*, y mueve las rotaciones de las placas tectónicas a la edad más antigua de la que se tenga información, esto se hace actualizando la edad que se quiere dentro de Rotations y después se visitan todas las placas tectónicas actualizando las rotaciones dentro de la función apply.

- osgGA::GUIEventAdapter::KEY_End. Realiza la operación contraria a la tecla supr, ya que mueve las rotaciones a la edad más reciente que

corresponde a la edad actual, realizando el mismo método de actualización de las rotaciones para las placas tectónicas.

- osgGA::GUIEventAdapter::KEY_Left. Provoca que el cuadro rojo se mueva 2 millones de años a la izquierda
- osgGA::GUIEventAdapter::KEY_Right. Realiza la operación contraria a KEY_Left ya que mueve 2 millones de años a la derecha.
- osgGA::GUIEventAdapter::KEY_Page_Down. Mueve la barra roja dentro de la barra de edades en intervalos de 10 millones de años hacia la izquierda.
- GUIEventAdapter::KEY_Page_Up. Mueve la barra 10 millones de años a la derecha.

La barra de edades también se puede manipular mediante el ratón, al hacer click izquierdo en algún lugar de la barra y el cuadro rojo que es el indicador de la edad en la que se encuentran las placas se mueve al punto donde se hizo click izquierdo, otra manera de manipulación es al hacer click izquierdo sobre el cuadro rojo de la barra y arrastrarlo hacia cualquier lado de la barra.

En el caso de los eventos que se generan por las interacciónes del usuario con el ratón se obtienen primero mediante el evento osgGA::GUIEventAdapter::PUSH, que es el que se activa cuando se aprieta algún botón del ratón o bien osgGA::GUIEventAdapter::DRAG que se activa cuando se presiona un botón y se mueve el ratón sin dejar de presionar el botón, al dar click se tiene que ver con cuál botón fue el que presionó, ya que se pueden capturar 3 botones.

- osgGA::GUIEventAdapter::LEFT_MOUSE_BUTTON: evento que sucede cuando se hace click en el botón izquierdo del ratón.
- osgGA::GUIEventAdapter::RIGHT_MOUSE_BUTTON: evento que sucede cuando se hace click en el botón derecho del ratón.
- osgGA::GUIEventAdapter::MIDDLE_MOUSE_BUTTON: evento que sucede cuando se hace click en el botón medio del ratón.

Para la interacción con la barra de edades sólo haremos algo cuando el botón que se presione sea el botón izquierdo tanto para cuando sólo se haga click o para cuando se arrastre, cada caso tiene un método que se encarga de esa acción: *keyPushhandler()* y *keyDragHandler()* respectivamente.

Para ambos métodos lo primero es crear un objeto de tipo: *osgUtil::Line SegmentIntersector* que se puede pensar como una línea recta que se crea a partir de las coordenadas x, y en donde se encuentre el ratón y que se extiende hasta el fondo de la escena, a este *osgUtil::LineSegmentIntersector* se le pasa a un osgUtil::IntersectionVisitor, este intersector funciona de manera semejante a la clase Visitor que se usó anteriormente ya que este visitor recorre todos los nodos colgados de la cámara, es decir todos los elementos que forman parte del grafo de escena, a estos nodos se les hace una transformación de tal forma que se pueden comparar con la línea que se creó y saber si esta línea los intersecta, en caso de que si toque algún nodo, la información de este es guardada dentro del *LineSegmentIntersector*.

Una vez que termina el *visitor* de verificar los nodos para verificar si existe intersección, el resultado es una lista de los nodos que si intersectan con la recta creada, estos nodos se consultan mediante un iterador para *LineSegmentIntersector*. A cada nodo dentro del iterador se le hace una comparación de su nombre para ver si coincide con los nombres de los elementos de la barra de edades o las placas tectónicas que se nombraron anteriormente.

Conociendo en que punto es intersectada la barra de edades o hacia que posición se arrastró el selector con el ratón, el selector se traslada a dicha posición mediante una matriz de traslación en x, lo cual automáticamente actualiza la rotación de las placas y el texto de la edad como se mencionó anteriormente al implementar el método *traverse()* para el objeto selector, el resutado de esto está en la siguiente figura.

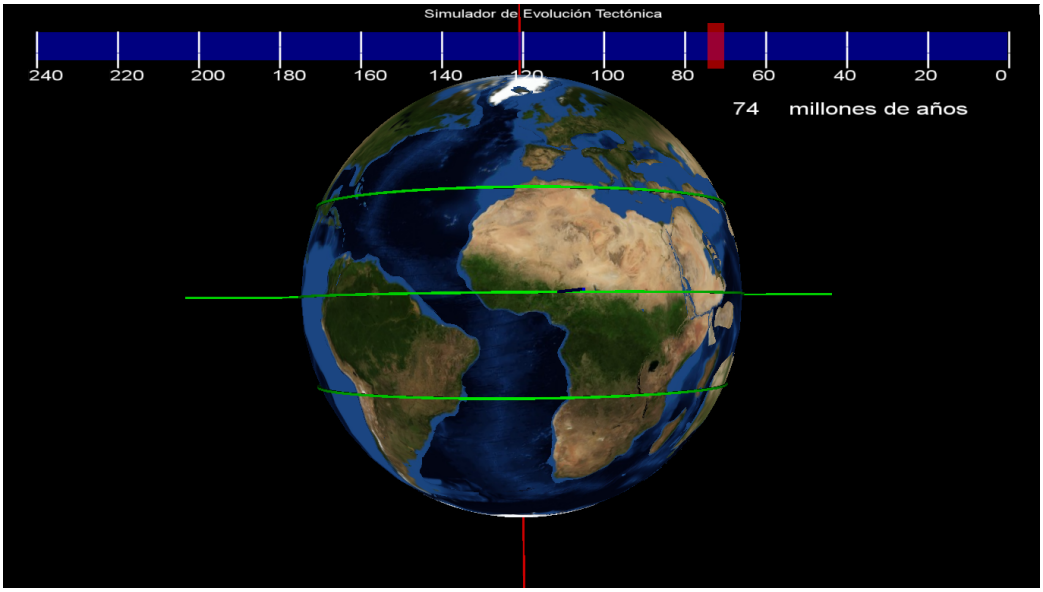

Figura 5.10: Movimiento de las placas tectónicas por medio de los eventos

5.5. Selección e Información de Placas Tectónicas

Para mostrar la información de las placas es necesario seleccionar alguna placa, e indicarle al usuario que ha seleccionado algo de alguna manera, por lo tanto se empezará por describir como es posible identificar la placa que el usuario desea seleccionar.

Como se mencionó anteriormente, el método para saber si se dio click sobre algún elemento gráfico de la pantalla es creando una línea partiendo del punto en donde se hizo click y de ahí se verifica si existe algún elemento que intersecte con la línea creada. En este caso lo que importa es saber si alguna placa tectónica ha sido intersectada, y en específico cual de todas ha sido, para esto el método de interesección utilizado regresa una lista de nombres de los elementos que intersectó, dentro de estos nombres se busca la palabra: "plate", ya que los nombres de las placas tectónicas contienen esta palabra con lo cual sabemos que una placa fue intersectada y se le pasa a la clase *Visitor* este nombre, ya que se mandará a visitar a todas las placas tectónicas para ir comparando sus nombres y encontrar la placa que fue

intersectada por su nombre.

Dentro de la clase *Visitor* una vez que se identificó la placa, el valor del color para la placa se modifica dentro de los shaders, con lo cual se iluminará de color amarillo y con esto se logra maneja la necesidad del usuario cuando quiere seleccionar alguna placa tectónica.

Una vez seleccionada la placa tectónica sólo falta mostrar información en la pantalla acerca de la placa, la información de las placas previamente se guardo dentro de un *std::map* en la cual por defecto para cada placa cargada se guardo el nombre de la misma, ya que se tiene una información por defecto se aplica otro método dentro del cual se actualiza los datos de las placas por información propia y mucho más completa que el simple nombre de la placa. Para mostrar estos datos se manda a llamar al método: *showStringPlate(std::string)* que pertenece a la clase de *GUIElements* donde se busca por la llave del mapeo, en este caso una cadena, la información y con los métodos anteriormente explicados de mostrar texto, se muestra la información correspondiente a esa llave que se proporcionó.

5.6. Animación

La animación de las placas tectónicas permite visualizar el movimiento de estas sin la necesidad de estar moviendo las placas manualmente, esto es útil ya que se puede usar para explicar algún tema en clase mientras las placas se mueven. Para lograr la animación se necesita hacer uso de las clases osg::AnimationPath y osg::AnimationPathCallback.

La clase *osg::AnimationPath* lo que permite es crear una animación sobre algún camino, en este caso será la animación sobre el selector de la barra de edades, ya que como se mencionó anteriormente el selector es una clase completa en donde se actualizan de manera automática las posiciones de las placas tectónicas y el texto de la edad actual al momento de dibujar el cuadro selector. Para crear dicho camino de animación se agrega al *osg::AnimationPath* puntos de control; cada punto de control consiste en un *osg::Vec3* que indica la traslación que se hará y un *osg::Quat* que indica la rotación que se quisiera hacer, en el caso de esta animación sólo se agrega un punto de control cada 10 millones de años en la barra haciendo una traslación sobre el eje x, y ninguna rotación. Además al *osg::AnimationPath* se le indica que se repetirá una y otra vez cuando acabe de recorrer todos los puntos de control desde el inicio.

Una vez que se tienen los puntos de control, el *osg::AnimationPath* se asigna al *osg::AnimationPathCallback*, este último es una llamada de actualización para la animación en el dibujado de cada cuadro de la escena que estará ligado al cuadro selector de la barra, por lo tanto una vez que el *osg::AnimationPathCallback* sea ligado al nodo del selector la animación comenzará a ejecutarse y solo se pausará o detendrá cuando el usuario presione la tecla P.

Capítulo 6

Resultados

Una vez que se termina la construcción se tiene un sistema funcionando con las caracteristicas que se habían pedido, para ver si el resultado es aplicable dentro del mundo real se organizó una reunión con algunos estudiantes y profesores de Ciencias de la Tierra para evaluar que les parecía el programa y ver la respuesta en el manejo del software con usuarios prototipo, esto es, los usuarios que principalmente usarán el programa. De dicha reunión se obtuvo lo siguiente:

6.1. Problemas:

- En un principio el color de selección de las placas tectónicas era rojo, sobre este tema los usuarios comentaron que el color rojo no ayudaba a percibir de manera clara la selección de alguna placa en particular ya que si se ponía la imagen del fondo oceánico termal era muy difícil ver la selección.
- Los usuarios al tratar de usar el programa tuvieron muchas dudas acerca de los controles con los cuales cuenta el simulador.
- Se detectó que al momento de activar la animación de las placas tectónicas el texto que indica los millones de años en los que se encuentran las placas no se actualizaba correctamente.
- Los usuarios mencionaron que sería conveniente que se mostraran las líneas de ecuador y los trópicos.

6.2. Soluciones:

- El color de la placa que se seleccione se cambio a amarillo con lo cual es más fácil determinar cual placa esta seleccionada.
- Se agregó un recuadro de ayuda al presionar la tecla H, con lo cual el usuario no tiene que estar revisando ningún manual externo o archivo de texto, lo que mejora la experiencia.
- Se encontró que el problema por el cual el texto de los millones de años no se actualizaba correctamente, era debido a que su actualización estaba ligada con los eventos del ratón, es decir, sólo se actualizaba cuando se arrastraba el ratón con el cuadro selector en la barra de edades o se hacia click en ella, como la animación se hace automáticamente el evento nunca era activado, la solución fue poner todas las actualizaciones dependientes del cuadro selector al momento de la actualizacion de dibujado en la clase *SliderMatrixTransform.*
- Se agregaron las líneas del ecuador y los trópicos en color verde.

Los usuarios coincidieron que la barra de edades era una manera simple y rápida por donde comenzar a manipular el simulador, esto hace que un usuario que nunca ha estado en contacto con el simulador no se quede parado sin poder hacer algo con el programa, lo cual puede considerarse una forma de inmersión a cierto nivel.

Otro punto que salió de esta sesión con los usuarios es que ellos mencionaron que podría ser una buena idea usarlo en las preparatorias como material de apoyo para las clases en donde se comente lo relacionado a las placas tectónicas.

El producto resultante con todas las características sugeridas por los usuarios y el resultado de la construcción del programa es mostrado en las siguientes figuras:

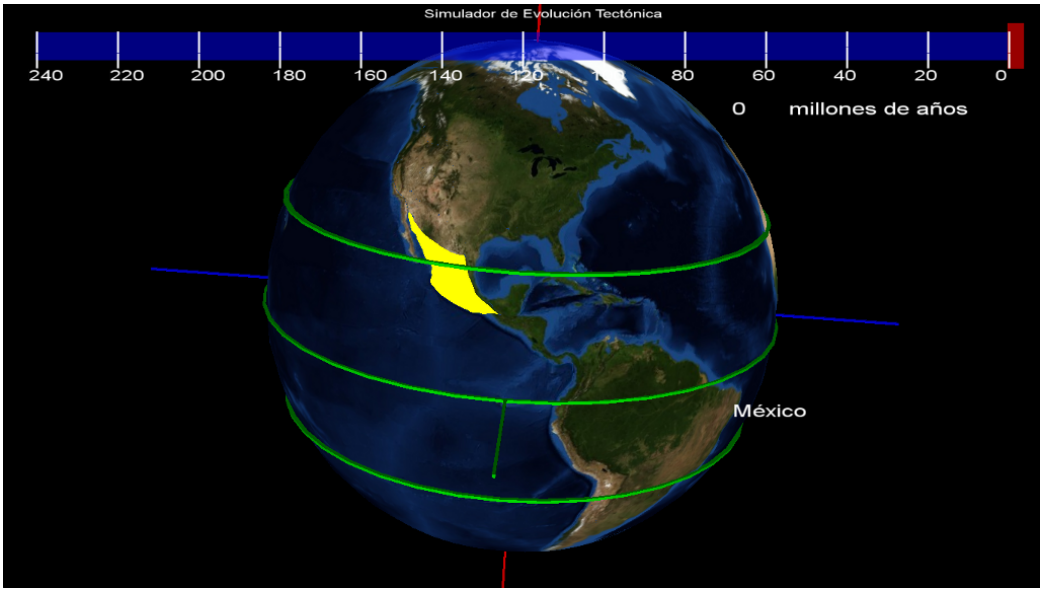

Figura 6.1: Selección de la placa tectónica correspondiente a México en la edad actual

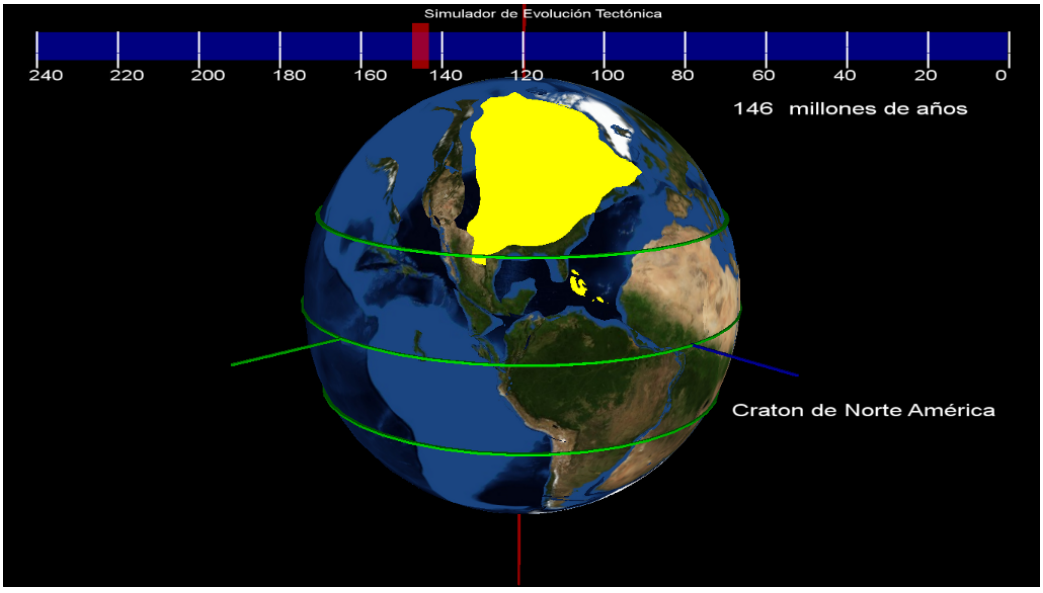

Figura 6.2: Simulador mostrando la posición de las placas hace 146 millones de años

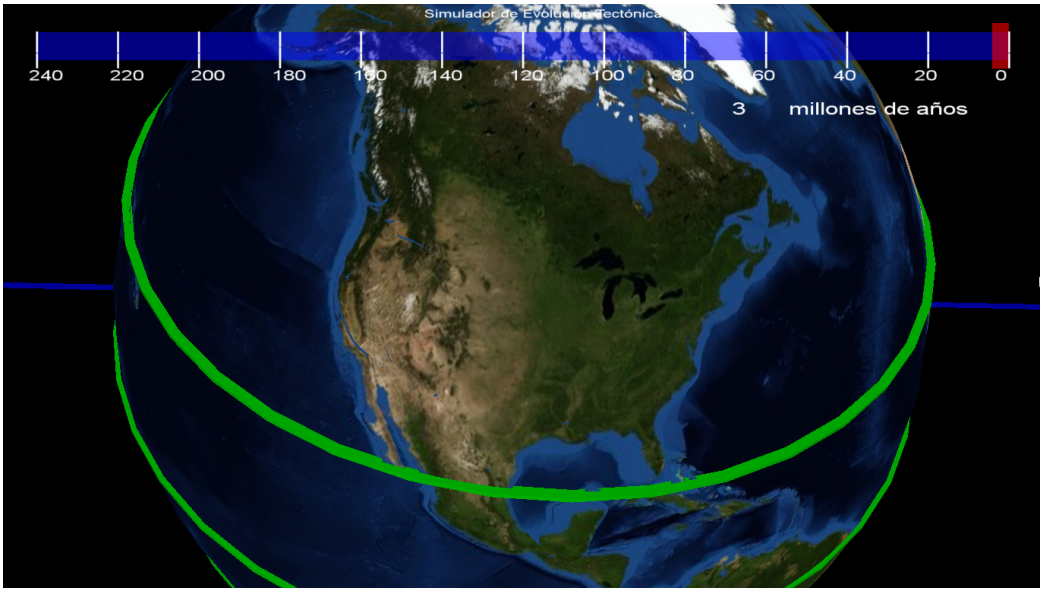

Figura 6.3: Globo terraqueo con zoom y cambio de texturas

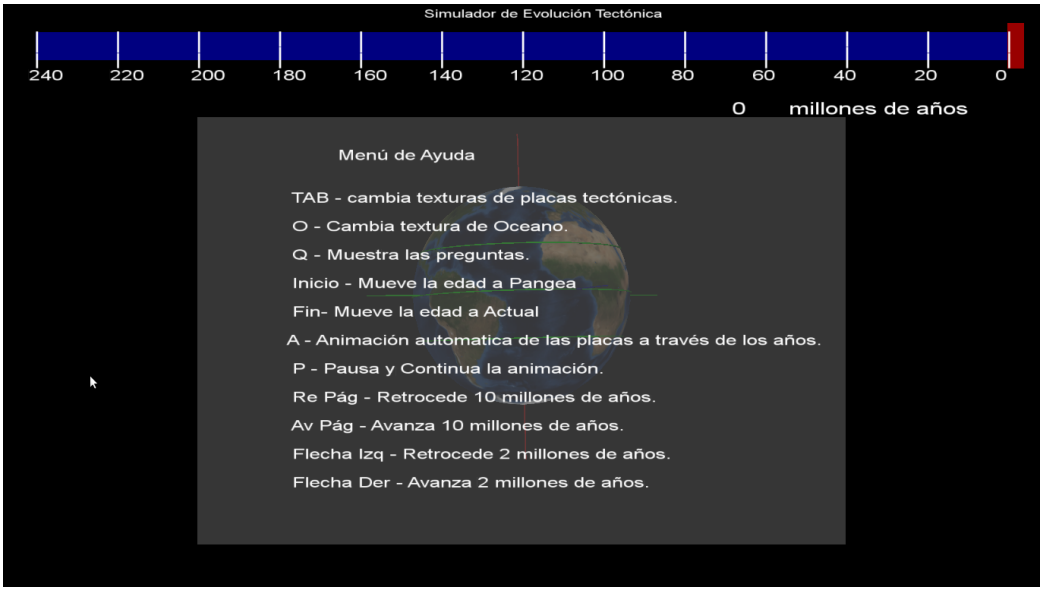

Figura 6.4: Cuadro de ayuda

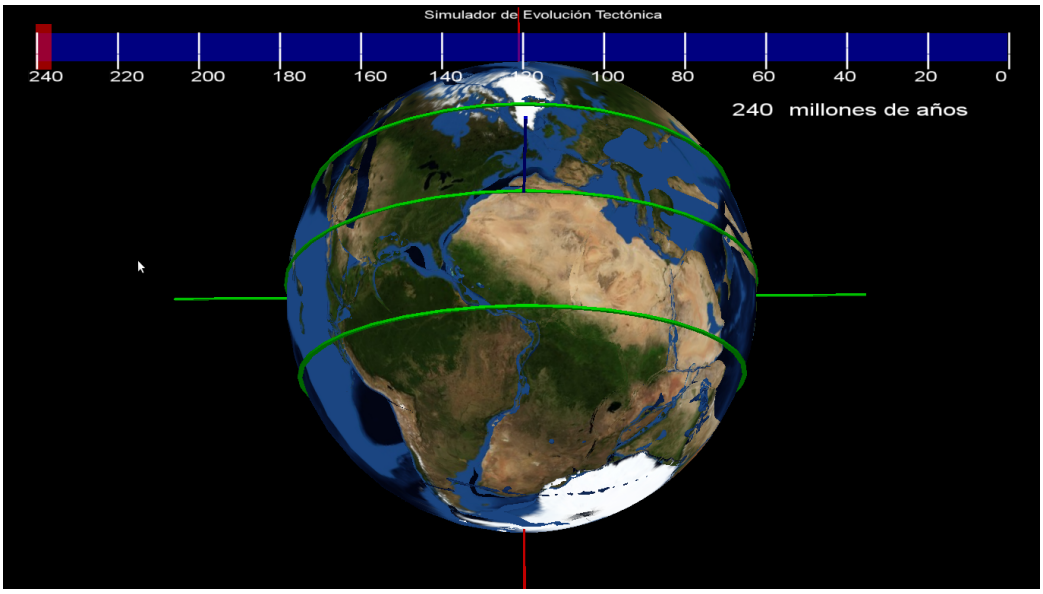

Figura 6.5: Posición de las placas hace 240 millones de años

Capítulo 7

Conclusiones y trabajo futuro

El poder de los sistemas computacionales en la actualidad se incrementa de forma considerable cada año, dando como resultado que las capacidades que tiene una computadora promedio en la actualidad sean 3 o veces mayor a una computadora de hace 4 años. Con este avance tan vertiginoso que se tiene dentro del mundo de los gráficos por computadora, hace que cada vez las computadoras sean capaces de desplegar efectos gráficos en tiempo real sorprendentes que hace un par de años solo se podían mostrar como un video, lo cual se nota en la mayoría de los videojuegos anteriores, ya que la introducción al videojuego lucía con gran detalle en todos los sentidos pero cuando se encontraba uno jugando esos detalles habían desaparecido debido a la gran complejidad, hoy en día este hecho ha cambiado y todo se procesa en tiempo real.

Lo mismo aplica para este simulador, ya que si bien no usa las últimas técnicas de graficación por computadora, la forma en la que se construye el software habrá evolucionado en un par de años, en los cuales puede que las bibliotecas gráficas se especialicen o modifiquen para aprovechar nuevas características de las tarjetas gráficas modificando su API. Otra vertiente diferente a ésta, puede ser el uso de la tecnología web usando gráficos tridimensionales, esto debido al impulso que ha tenido ultimamente *WebGL*, esta tecnología permite tener la capacidad de ver programas de *OpenGL* dentro de un navegador web como *Firefox* y *Chrome*, todo esto mediante el lenguaje JavaScript, y dado que el *framework OpenSceneGraph* se basa en *OpenGL* puede que en un período no muy lejano se tenga este framework funcionando dentro de *WebGL*; lo cual hará que muchos programas incluido éste sobrepasen la barrera de la multiplataforma o la distrubución del software ya que

todo está en internet y sólo será necesario un navegador para poder acceder al programa.

Actualmente para que un software sea usado por una gran cantidad de personas, es necesario que se vuelva multiplataforma, es decir, que se ejecute tanto en sistemas *Linux* como *Windows* y *Mac OS*, incluso hasta dispositivos móviles y tabletas que están teniendo un gran auge. Durante el desarrollo de este proyecto me di cuenta de la dificultad que ésto conlleva, ya que no es suficiente terminar el programa, que funcione correctamente y que lo puedas instalar de forma manual en un entorno local pequeño, como puede ser en las computadoras de algunos profesores de Ciencias de la Tierra e incluso algunos alumnos, o en la sala *Ixtli* (como estaba pensado inicialmente) en donde solo fuera usado este software, sino que si la nueva intención es llevarlo como un programa simple dentro del nivel de preparatoria, se vuelve caotico hacerlo de forma manual. Un trabajo a futuro para este proyecto es añadir la forma de distribuirlo ampliamente, ya sea utilizando instaladores o *WebGL*, ya que como se menciona la forma inicial de uso de este software fue la sala *Ixtli*, lo cual excluye en este proyecto la distribución a multiples computadoras.

Con la realización de este trabajo se hace más que evidente que actualmente ningún programa está concluido al 100 %, no se puede crear algo y dejarse ahí para su uso sin ningún tipo de actualización, el destino de ese tipo de software será su muerte, para que un programa esté vigente tiene que actualizarse por dos razones que considero las más importantes:

- Los usuarios al manejarlo aportan ideas que pueden parecer simples para uno como el que programa, pero ese simple cambio puede significar la diferencia entre una adopción aceptable del software o su abandono.

- El uso de nuevas tecnologias que hagan la experiencia de usuario más agradable ya sea por mejora de efectos visuales o por facilidad de acceso desde cualquier dispositivo.

El error más grande que se puede cometer al estar desarrollando un programa es creer que el programa está totalmente completo o que lo que se hace nunca va a estar lo suficientemente listo para ser usado.

Índice de figuras

1.1. Evidencia de restos fósiles en varios lugar del mundo que hoy en dia se encuentran separados (Kious and Tilling [5]). 9
1.2. Pangea y su fragmentación a través de los años (Kious and Tilling [5]). 10
1.3. cadena montañosa a lo largo del fondo oceánico (usgs.gov) . . 12
1.4. Variaciones de polaridad magnética en la roca basáltica(Kious and Tilling [5]). 13
1.5. Convergencia de una placa continental con una placa oceánica (Kious and Tilling [5]). 14
1.6. Convergencia de dos placas oceánicas (Kious and Tilling [5]). . 15
1.7. Convergencia de dos placas continenales (Kious and Tilling [5]). 15
1.8. Límites transformantes 16
1.9. Triángulos semejantes formados por la traslación de la interpolación esférica (Lengyel [6]) 27

3.1. Polígonos de placas tectónicas, Universidad de Texas (Sierra et al. [8]) 33
3.2. Polígonos de placas tectónicas, Scotese 2009 (Sierra et al. [8]) . 34
3.3. Polígonos de placas tectónicas, EarthByte 2009 (Sierra et al. [8]) 34
3.4. Resultado de la triangularización de las placas tectónicas, usando los datos de los polígonos de Scotese. (Sierra et al. [8]) . . . 36

4.1. Diagrama de Casos de Uso 40
4.2. Esquema de un Modelo Vista Controlador típico 43
4.3. Diagrama de Paquetes para el simulador de placas tectónicas . 44

5.1. Primitivas Gráficas (Lengyel [6]). 46

5.2. Comunicación efectuada entre el CPU y la GPU (Lengyel [6]). 48
5.3. Grafo de escena general 52
5.4. Diagrama de Nodos del Globo Terráqueo 54
5.5. Imagen de la Tierra con la textura del fondo oceánico 57
5.6. Imagen de la Tierra con la textura térmica del océano 58
5.7. Diagrama de los nodos de los elementos de interface gráfica . . 59
5.8. Elementos de la interface gráfica agregados al programa 62
5.9. Placas tectónicas cargadas y agregadas al grafo de escena . . . 67
5.10. Movimiento de las placas tectónicas por medio de los eventos . 71

6.1. Selección de la placa tectónica correspondiente a México en la edad actual 76
6.2. Simulador mostrando la posición de las placas hace 146 millones de años 76
6.3. Globo terraqueo con zoom y cambio de texturas 77
6.4. Cuadro de ayuda 77
6.5. Posición de las placas hace 240 millones de años 78

Bibliografía

[1] Ana María Soler Arechalde. Tectónica de placas, 2011.

[2] B. Greiner. Euler rotations in plate-tectonic reconstructions. page 8, 1999.

[3] Mario A. Gutiérrez, Frédéric Vexo, and Daniel Thalmann. *Stepping Into Virtual Reality*. Springer, 2008.

[4] Guadalupe Ibargüengoitia and Hanna Oktaba. *Ingeniería de Software Pragmática*. 2010.

[5] W. Jacquelyne Kious and Robert I. Tilling. *The Dynamic Earth: The Story of Plate Tectonics*. 1996.

[6] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics*. Course Technology, 2012.

[7] Dave Shreiner. *OpenGL Programming Guide, Seventh Edition*. Addison-Wesley, 2010.

[8] Alejandro Aguilar Sierra, Ana María Soler Arechalde, Félix García Gutierrez, and René Garduño López. Evolución tectónica del planeta tierra. Technical report, Ciencias de la Atmosfera, 2011.

[9] Rui Wang and Xuelei Qian. *OpenSceneGraph 3.0: Beginner's Guide*. PACKT, 2010.