UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA INFORMÀTICA

UNIVERISTAT POLITÈCNICA DE VALÈNCIA

Herramientas de análisis y visualización de nubes de puntos (LiDAR) para la aplicación PointCloudViz

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: David Guerrero Cabrerizo

Tutor: Javier Lluch Crespo

2015 - 2016

RESUMEN

En este proyecto se crean una serie de herramientas para la aplicación de análisis de modelos 3D llamada PointCloudViz. Dichas herramientas proporcionan a los usuarios nuevas funcionalidades con un manejo sencillo y fluido de utilizar. Para ello se implementará una *API* que hará posible su integración desde otras aplicaciones similares.

RESUM

En aquest projecte es creen un conjunt de ferramentes per a l'aplicació de anàlisis de models nomenada PointCloudViz. Aquestes ferramentes proporcionen als usuaris noves funcionalitats de maneig senzill i fluid de utilitzar. Per a això s'implementarà una *API* que farà possible la seua integració des d'altres aplicacions similars.

ABSTRACT

In this project a couple of tools are created for the 3D models analysis application called PointCloudViz. These tools provide users new functionalities with simple and fluid handling. For this purpose, an *API* will be implemented that will enable integration from other similar applications.

AGRADECIMIENTOS

A mis padres y hermana principalmente, por ayudarme con el trabajo y con toda la carrera, sin ellos hubiera sido imposible.

A mi tutor, Javier Lluch Crespo por ofrecerme hacer este trabajo, que me gusto más que ningún otro.

Al equipo de Mirage Tech, por toda la ayuda que me han ofrecido de principio a fin para iniciarme en el mundillo y darme todo el soporte que me ha hecho falta.

Índice

1. Introducción

Los primeros intentos de representar el mundo físico y dejarlo plasmado en un medio digital tienen origen a principios de siglo. Unas de las aplicaciones hoy en día más famosas en cuanto a esta área se refiere, son *Google Maps,* que empezó a manos de la empresa *Where2*, y *Google Earth*, que fue diseñada por la compañía *Keyhole*. Ambas compañías fueron adquiridas por Google en el 2004, y sus productos empezaron a ser parte de la mundialmente famosa compañía.

Sin embargo, viendo como la tecnología avanzaba desorbitadamente, el diseño que ofrecía Google en sus productos parecía insuficiente, ya que, en definitiva, lo único que mostraban eran un conjunto de fotografías en 2 dimensiones. Mirage-Tech fue una empresa nacida en 2009 que decidió cambiar el tipo de modelo ofrecido en este tipo de aplicaciones, y empezó a construir una aplicación, a la que llamaron PointCloudViz.

Esta aplicación permitía mostrar un modelo en 3 dimensiones mediante datos LiDAR, y que se visualizaban en forma de lo que a partir de ahora llamaremos nube de puntos, y empezaron a construir su nueva aplicación en un entorno llamado OSG, que les proporcionó las herramientas necesarias para realizar su cometido. Con este nuevo concepto, una imagen de un campo de fútbol que podíamos observar mediante Google Maps en 2 Dimensiones, podía ser vista en 3 Dimensiones en nuestra aplicación mediante un conjunto muy grande de puntos ubicados en el espacio de coordenadas del programa, de los que además de posición en el espacio, podemos saber su color, y con esta información y sabiendo de donde se extrae, podremos clasificar cada objeto según el material del que este hecho. Los usos de esta aplicación son diversos: podemos desde visualizar un terreno montañoso, del que luego podremos extraer información real como altura máxima, área... hasta analizar un bosque, para saber el grado de daños de un incendio.

Este trabajo parte con esta aplicación ya construida, y pretende añadir una serie de funcionalidades que mejorarán la experiencia del usuario al usar esta aplicación, o cualquier otra que pueda integrarla mediante la API que vamos a proporcionar. Proporcionaremos al usuario herramientas para el análisis de los modelos previamente cargados, con las que obtendremos una información u otra dependiendo de la herramienta utilizada. Dispondremos herramientas para localización local de los puntos seleccionados con el puntero del ratón, y relativos a las coordenadas del modelo cargado, medición de múltiples distancias reales entre puntos del modelo, y selección de áreas de influencia con degradado de colores.

Comenzaremos planteándonos una serie de objetivos que a medida que realicemos el trabajo iremos cumpliendo. Asimismo, haremos un recorrido de todas las herramientas ya existentes que nos pueden ser de ayuda para realizar el proyecto, detallaremos las más importantes con características propias que nos han sido útiles, y finalmente explicaremos por cuáles de ellas nos decantamos y por qué. Más adelante presentaremos las herramientas que hemos realizado, y haremos un profundo recorrido por cada una de ellas: empezaremos por su definición básica, mostrando los diagramas más importantes, con lo que podremos ver cómo interactúan las herramientas entre ellas; ejemplos de uso...

Después de dicha presentación, tomaremos esta como referencia para definir los aspectos más técnicos del proyecto. Comenzaremos detallando como hemos usado las herramientas ya existentes previamente mencionadas, así como la manera en que hemos trabajado con ellas, como han sido integradas... Y finalmente detallaremos los procesos internos de la aplicación, aritmética para los cálculos con vectores, interacción de las herramientas con el ratón... Éste será básicamente el punto en el que daremos los detalles más profundos de la programación de las herramientas.

A continuación mostraremos unas imágenes de cada una de las herramientas que hemos diseñado. Como han sido diseñadas para ser integradas, las mostraremos inicialmente sobre el entorno con el que hemos trabajado, viendo cada funcionalidad propia, y finalmente podremos ver el resultado de integrar dichas herramientas en PointCloudViz, que como hemos dicho previamente es uno de los pilares de este trabajo.

Finalizaremos este documento mostrando nuestras conclusiones, cumplimiento de objetivos, dificultad del proyecto, y daremos a conocer una serie de mejoras que podemos llevar a cabo en un futuro con el objetivo de mejorar aún más la experiencia del usuario.

2. Objetivos

Asimismo, nos marcaremos ciertos objetivos a cumplir a lo largo del desarrollo de este trabajo. El más esencial de ellos será cubrir una necesidad del usuario final, la cual ha sido la que ha inspirado a realizar este trabajo, y que básicamente se solucionará con nuestro producto final. Esta meta la alcanzaremos mediante una serie de submetas, que se verán alcanzadas a medida que cada una de las 4 herramientas que diseñaremos por separado se vayan completando, y que más tarde uniremos para crear una API que cualquier persona que realice modelados 3D podrá usar para realizar análisis en tiempo real de los modelos que hayan cargado previamente.

La primera de ellas, y la más simple, aunque no sencilla, nos servirá para obtener información local de cada punto del espacio de los que se compone el modelo de nube de puntos. Esto nos servirá para extraer las coordenadas de cualquier punto respecto al origen de coordenadas del modelo, así como el color del material que ha sido seleccionado. Además, se podrán deducir el material del objeto señalado, realizando una serie de extrapolaciones, siendo clasificado mediante las variables color obtenido y el espacio del que se realizó el modelo.

La segunda es consiguiente a la primera; obteniendo la información local de una serie sucesiva de puntos, podemos formar una poli línea, y mediante operaciones aritméticas podremos calcular distancias reales entre los sucesivos puntos, y mostraremos también la distancia total de las distintas líneas dibujadas.

Las siguientes herramientas dejan atrás a este primer grupo de utilidades para obtener información espacial del modelo, para dejar paso al uso de buffers, con los que podremos obtener distinta información. Una de ellas nos facilitará el conocimiento de áreas de influencia respecto a un punto central. Este mostrará un degradado entre dos colores, siendo el primer color el de los puntos más cercanos y el segundo el de los más lejanos.

Si bien esta anterior herramienta nos proporciona información respecto a un punto, la siguiente de ellas y última nos facilitará información respecto a varios centros; es decir, obtendremos una serie de buffers sucesivos que se solaparán entre sí.

Para todas ellas deberé aplicar mis conocimientos previos, que más adelante nombraremos, y que complementaré con unos nuevos que serán útiles para el futuro.

Cabe destacar que uno de los objetivos a partir del cual nace el trabajo, es de añadir funcionalidades a la aplicación ya nombrada PointCloudViz. Una vez acabadas dichas herramientas, se procedería a integrarlas con un previo lavado de cara, con el que dichas herramientas quedarán completamente adaptadas a dicha aplicación. A su vez conseguiremos cumplir el primer objetivo, la satisfacción del usuario final, que en este caso serán los usuarios existentes de PointCloudViz y que disfrutarán de nuevas opciones cuando estén utilizándola.

3. Antecedentes

Con el fin de familiarizarnos con este software, presentaremos a continuación una serie de componentes técnicos que hemos utilizado y que han sido claves para el desarrollo de éste. Además expondremos por qué hemos decidido implementar esta nueva solución y no hemos recurrido a una existente. Los elementos más representativos han sido por orden:

- Shaders
- OSG
- LiDAR
- PointCloudViz

A continuación entraremos en detalle con cada uno de ellos.

3.1 Shaders

Los Shaders son unos fragmentos de código independientes al programa principal, que nos permiten interactuar con la GPU o tarjeta gráfica, y que pueden ser escritas en varios lenguajes de programación, pero nosotros utilizaremos un lenguaje basado en el lenguaje de programación C llamado GLSL [11] [12]. Muy a grosso modo, estos nos permiten crear efectos especiales, bien sean fuegos, nieblas, iluminación, así como crear texturas de una manera más realistas jugando con luces y sombras[5].

En este trabajo emplearemos dos tipos de Shaders:

- Vertex Shader, que aplican el programa punto por punto, por tanto más delicado
- Fragment Shader, que lo realizan por fragmentos de imagen.

En la Figura 1 tenemos un ejemplo como queda un modelo antes y después de aplicar un Vertex Shader, en el que aplicamos una textura punto a punto.

Figura 1: Rostro antes y después de aplicar un Vertex Shader

3.2 OSG

Otro antecedente clave de este trabajo es OSG, cuyo desarrollo empezó en 1998 en manos de Don Burns y Robert Osfield que se unió al proyecto en 1999. Básicamente éste nos proporciona una estructura en forma de grafos, para designar las dependencias entre componentes, y haremos uso de la gran cantidad de librerías que nos provee para realizar todo tipo de operaciones [1] [2]:

- Aritmética simple para cálculos entre vectores.
- Creación de formas geométricas.
- Manipulación de cámaras entre otras.
- Manejo de Shaders anteriormente explicados.

A continuación presentaremos algún ejemplo de aplicación diseñada con OSG.

3.2.1 FlightGear

FlightGear es un simulador de vuelo multiplataforma y libre. Surgió en 1997 como alternativa importante frente a los simuladores de vuelo comerciales, y actualmente va por su versión v2016.2. Como ya sabemos, está construido con OSG, lo que le proporciona suficiente amplitud para el diseño de gráficos, además de una funcionalidad muy avanzada [7]. Veamos un ejemplo en la Figura 2.

Figura 2: Captura de Pantalla de FlightGear

3.2.2 Capaware

Capaware comenzó en 2007 como una iniciativa del Gobierno de Canarias para su uso como visor 3D propio en temas de gestión de emergencias, mediante la interacción con terrenos virtuales 3D, que disponen de una precisión cartográfica. También trabaja con OSG, con el que se logran tasas de refresco elevadas y una fluidez del movimiento necesaria para este tipo de herramienta [8]. Veamos una imagen de Capaware en la Figura 3.

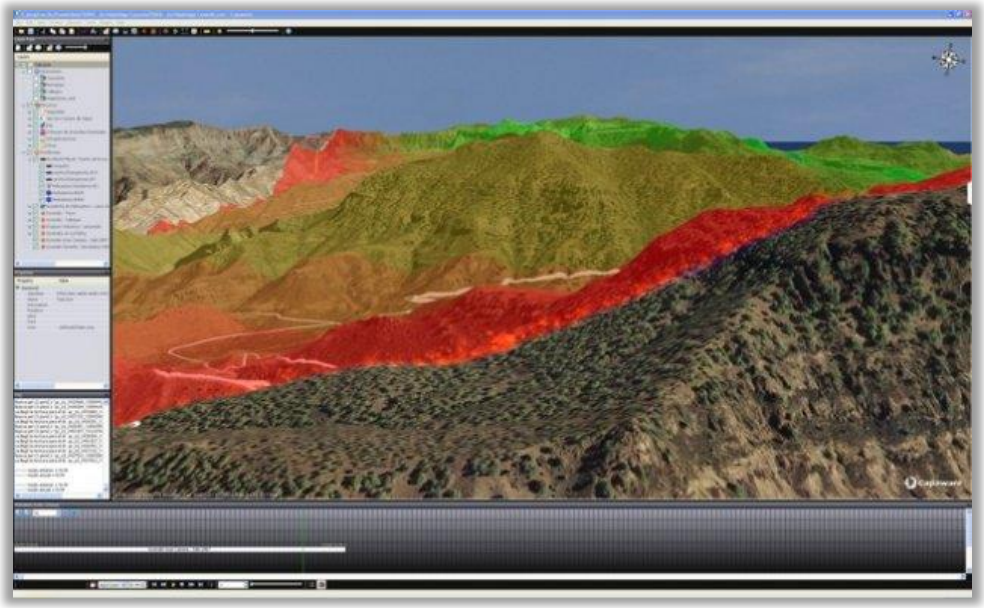


Figura 3: Entorno Virtual 3D de Capaware

3.3 LiDAR

El precedente clave para la estructuración de los datos en los modelos 3D es LiDAR (un acrónimo del inglés *Light Detection and Ranging* o *Laser Imaging Detection and Ranging*) es una tecnología que nos permite obtener la distancia desde un foco emisor a un objeto o superficie utilizando un haz láser pulsado. Dicha distancia es determinada calculando el tiempo de retraso desde que el pulso se emite, hasta que llega desde el objeto en el que el haz ha sido reflejado, y lo haremos tantas veces como veamos necesario como para acumular una serie de puntos que puedan formar el modelo 3D. Una curiosidad es que su principio es bastante similar al de los SONAR, con la diferencia de que este lanza ondas sonoras, no de luz [9]. En la Figura 4 podemos ver como es el dispositivo que hace los escaneos.

Figura 4: Dispositivo LiDAR Leica

Asimismo disponemos de 4 tipos de escaneado para obtener datos LiDAR:

- Líneas. Produce líneas paralelas en el terreno como patrón de escaneado. El inconveniente principal de este sistema es que al girar el espejo en una sola dirección no siempre tenemos mediciones.
- Zigzag. En este caso el espejo es rotatorio en dos sentidos (ida y vuelta). Produce líneas en zigzag como patrón de escaneado.

- De fibra óptica. Este sistema produce una huella en forma de una especie de circunferencias solapadas.
- Elíptico: En este caso el haz láser es desviado por dos espejos que producen un patrón de escaneado en forma de elipse.

3.4 PointCloudViz

Son muchos los precedentes de este trabajo, pero PointCloudViz es el más importante, ya que las herramientas realizadas fueron inicialmente pensadas para ser añadidas a su interfaz, además de poder ser futura integración en aplicaciones externas. PointCloudViz es una aplicación que fue diseñada por Mirage Technologies SL en 2009, y nos permite analizar modelos tridimensionales previamente cargados.

Además nos permite interactuar con ellos mediante una serie de herramientas que nos proporciona:

- Clasificación de materiales, en bosque, agua, montaña…
- Clasificación de objetos según su altura.
- Anotaciones particulares en tiempo real.

Para construir estos modelos, la aplicación convierte datos LiDAR, previamente explicados, en una nube de puntos en el sistema de coordenadas tridimensional que constituyen objetos sólidos, en nuestro caso, terrenos [10]. Podemos ver unos ejemplos de la aplicación PointCloudViz en las siguientes imágenes.

La Figura 5 nos muestra un paisaje que hemos cargado en la aplicación.

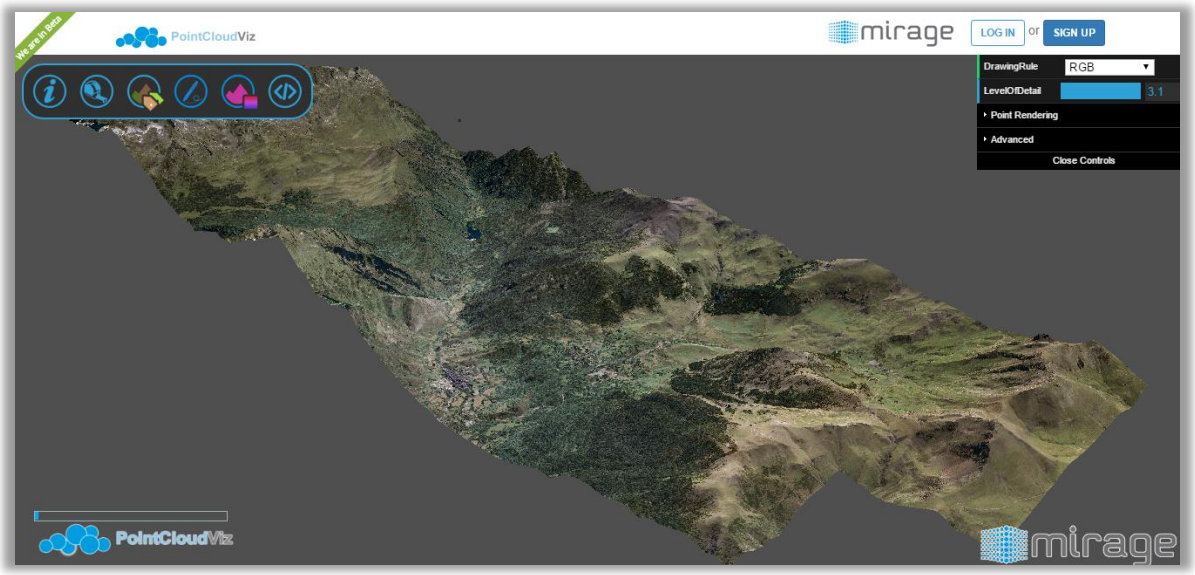


Figura 5: Vista general de Benasque y Cerler

Si hacemos un poco de zoom, será más evidente el cúmulo o nube de puntos que forman el modelo. Podemos ver un ejemplo de ello en la Figura 6.

Figura 6: Vista más cercana de la Isla Esperanza

A continuación, en la Figura 7 podemos observar un terreno en el que hemos clasificado los puntos correspondientes a suelo según su elevación, desde colores fríos para alturas bajas, a colores cálidos para alturas más altas.

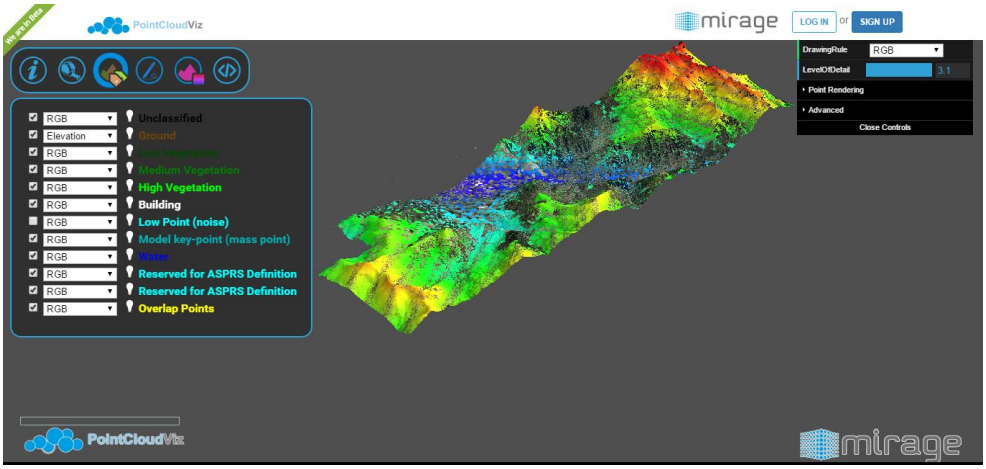


Figura 7: Benasque y Cerler clasificado según la elevación

3.5 Conclusión

Hasta ahora hemos analizado todos los antecedentes tomados para la realización de este proyecto, aunque en realidad existen alternativas a algunos de ellos, como por ejemplo a OSG. Otros motores gráficos que podrían habernos servido de base en el desarrollo de este proyecto son OGRE 3D, Source y BLAM! ¿Por qué nos hemos decantado entonces por OSG? Pues tenemos diversas razones:

- Su uso es gratuito, no requiere licencias de pago ni nada similar.
- Es de fácil instalación [4].
- Es de código abierto, por lo que podemos saber más acerca de su funcionamiento interno [3].
- Está escrito en C++, lenguaje con el que estoy bastante familiarizado.
- Me ha resultado fácil encontrar información sobre OSG, así como foros en los que podía preguntar e informarme, y una gran cantidad de ejemplos, cosa que con los otros no (a excepción de OGRE3D).
- OSG trabaja con Shaders, por tanto su integración se hace sencilla.

- La aplicación que hemos tomado como base es la ya nombrada PointCloudViz, y esta usa OSG como motor gráfico, además de realizar conversiones de los datos LiDAR, previamente nombrados.
- En cuanto a los Shaders, existen páginas web donde encontrar muchos ejemplos [6].

En definitiva, aunque hemos tenido otras opciones válidas para el desarrollo de este proyecto, las hemos descartado, y en realidad se hizo muy fácil elegir OSG por la sencilla razón de que si la aplicación PointCloudViz ya estaba basada en él, para realizar una más fácil integración de nuestras herramientas en ésta era mucho más sencillo utilizar las mismas librerías.

4. Análisis y Diseño

Hemos construido una serie de herramientas que más tarde podemos integrar a otras aplicaciones del mismo tipo que PointCloudViz. A continuación detallaremos como los hemos construido, como hemos ligado cada una de las clases, y como podremos integrar posiblemente estas herramientas en otra aplicación.

4.1 Clase osgATViewer

Inicialmente disponemos de una clase principal a la que llamamos osgATViewer (Analysis Tool Viewer), desde la que crearemos:

- Objetos de las clases (fuera de esta clase, cosa que luego aclaramos) que implementan cada una de las herramientas, y que más tarde detallaremos.
- Un objeto de otra clase que será la encargada de manejar todos los eventos de la aplicación y externos a él, para así poder acceder a las distintas herramientas y explotar cada una de sus características.
- Y un objeto de otra clase para manejar los eventos de manipulación de cámara; los comportamientos que emplea la aplicación para funcionar correctamente los explicaremos más adelante.

Éste será básicamente el comportamiento a introducir en la aplicación externa cuando vayamos integrar las herramientas objeto de este trabajo. Podemos ver un diagrama en UML de lo que acabamos de describir en la Figura 8.

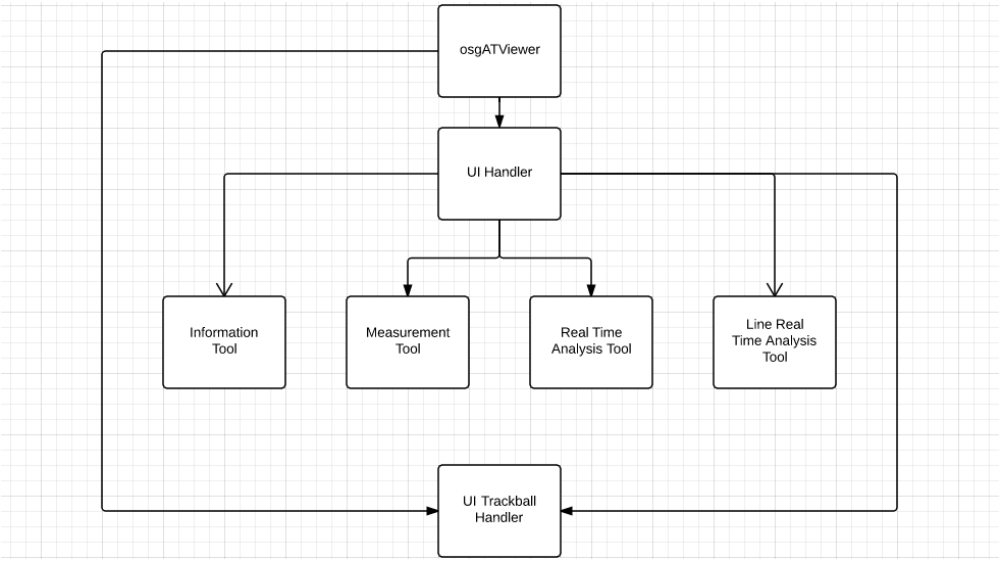


Figura 8: Diagrama de clases general

4.2 Clase UIHandler

La clase UI Handler será descendiente de la clase GUIEventHandler, con lo cual todos los eventos a manejar que no sean propios de cada herramienta, es decir, que no activen funciones propias de la misma... se manejarán con dicha súper clase. El resto de eventos los manejaremos activando el manejador de eventos de la clase con la que estemos trabajando; esto quedará más claro con las posteriores explicaciones. Como ya hemos dicho, la clase principal contiene un objeto de la clase UI Handler, así como otro de la clase UI Trackball Handler. Pues a su vez este objeto de UI Handler tiene acceso al segundo, por lo que así podremos activar el evento correspondiente según la tecla pulsada, desde el manejo de la cámara, al manejo de la herramienta activa, teniendo siempre como máximo un tipo de evento activo, cámara o herramienta, y una herramienta activa como máximo. Esto nos permitirá que los eventos sean correctamente manejados, sin dar opción al sistema a confundir qué eventos van asociados a qué herramienta. Lo más importante es que en esta clase solo vamos a implementar el cambio de manejo de cámara al de herramientas, es decir, mediante una tecla pasaremos de mover la cámara a usar cualquiera de las herramientas. El manejo de la cámara se hace desde la clase UI Trackball Handler y el de cada herramienta desde la clase en la que se implementa dicha herramienta, es decir, que cada clase tendrá una función para manejar sus propios eventos y la clase UI Handler hará de puente para que estas se ejecuten. Podemos ver el funcionamiento en la Figura 9.

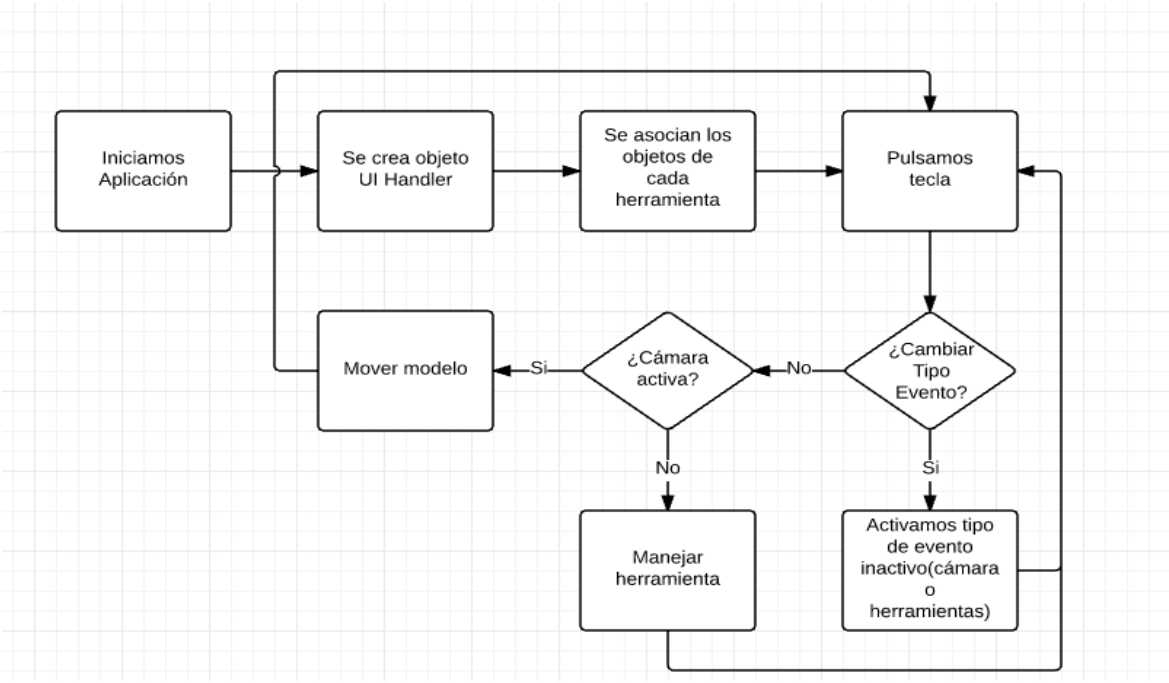


Figura 9: Funcionamiento general del programa

4.3 Breve descripción de las herramientas

A continuación vamos a explicar el funcionamiento general de cada clase que implementa una herramienta. Son 4, Information Tool, Measurement Tool, Real Time Analysis, y Line Real Time Analysis. Todas ellas dispondrán serán clases hijas de la clase PointCloudVizTool, en la que vamos a disponer básicamente de una variable para saber si una herramienta está activa o no. Luego, en cada una de las clases dispondremos de una variable para saber el estado de la herramienta, el método de manejo específico de cada clase, y otras diferentes funciones según la clase.

4.4 Information Tool

En la primera de ellas, Information Tool, disponemos de un manejador que esperará a que pulsemos:

- El botón izquierdo del ratón para situar un marcador en el modelo, y en el que podremos ver información local del punto, tanto las coordenadas como el color de ese punto.
- ESC cuando queramos borrar el resultado, pulsar para borrar el último punto dibujado, y de nuevo ESC para borrar todo. Aunque el funcionamiento interno lo detallaremos más adelante, podemos echar un vistazo a la Figura 10 para ver el funcionamiento gráficamente

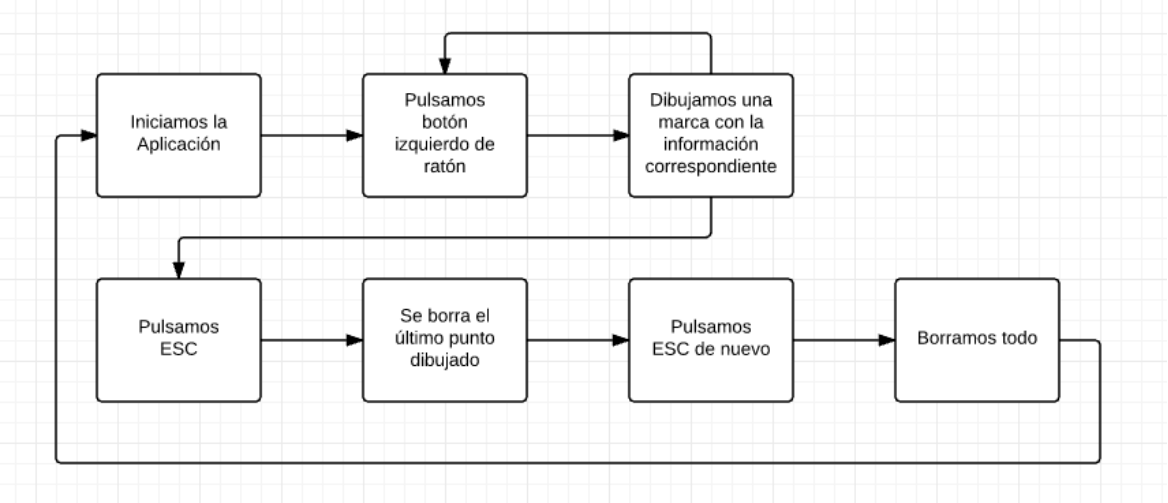


Figura 10: Funcionamiento básico de la Herramienta de Información

4.5 Measurement Tool

La segunda herramienta de la que disponemos es la de medidas. Básicamente hace lo mismo que la anterior, espera a que pulsemos el botón, y una vez pulsado se crean dos puntos, uno fijo y otro móvil con el que podremos medir distancias así como ver las deltas expresadas en (x,y,z) entre ambos puntos. A continuación listamos sus utidades:

- Podremos medir tantas distancias como queramos, y además cada distancia se verá reflejada en la mitad de cada arista entre cada dos puntos.

- Además habrá un cálculo de la suma de todas las distancias y de todas las deltas en la parte inferior izquierda.
- Al igual que en la herramienta anterior, podemos usar el botón ESC para borrar la última arista, y todas si lo pulsamos dos veces.

En la Figura 11 tenemos un diagrama con todas estas cosas que hemos explicado.

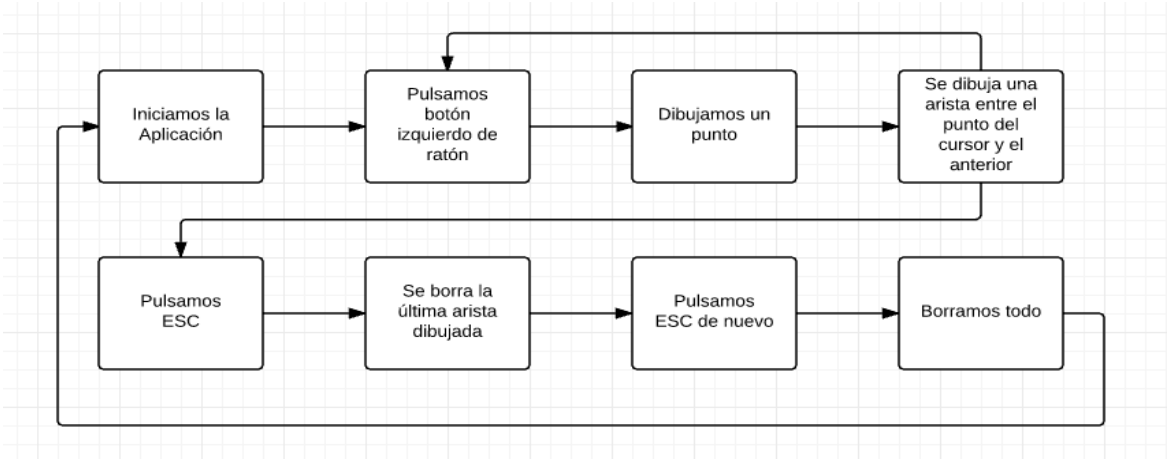


Figura 11: Funcionamiento básico de la Herramienta de Medidas

4.6 Buffer Tools

Disponemos de dos herramientas más, similares entre sí. Ambas utilizan Shaders, cuyo significado hemos aclarado antes, y los utilizamos para analizar áreas de influencia en base a un radio mediante un degradado de color desde rojo hasta azul. En la primera de las herramientas, cuando pulsemos el botón izquierdo del ratón, el programa nos dibuja un círculo con un degradado de colores desde rojo en el centro a azul en los bordes, y con ello vemos que partes están más cerca con los colores cálidos, y más lejanas con los fríos. Sin embargo, en la segunda herramienta dispondremos, como en la primera, también de dos puntos cada vez que pulsamos el botón izquierdo del ratón, y así podemos analizar áreas de influencia de las líneas dibujadas. Más tarde detallaremos como se han implementado ambas clases, así como el funcionamiento de los Shader y ver algún resultado visual.

En las Figuras 12 y 13 podemos ver los diagramas que representan todo esto.

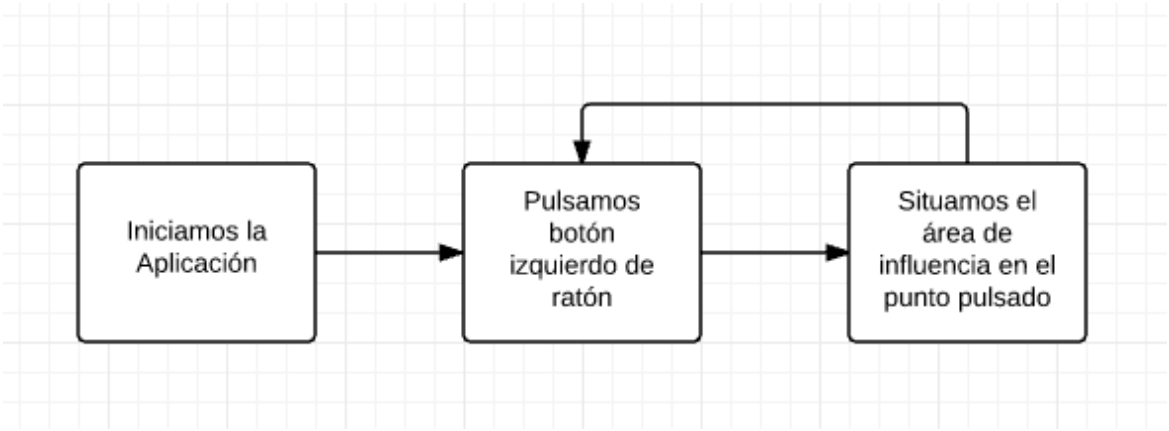


Figura 12: Funcionamiento básico de la Herramienta de Buffer 3D

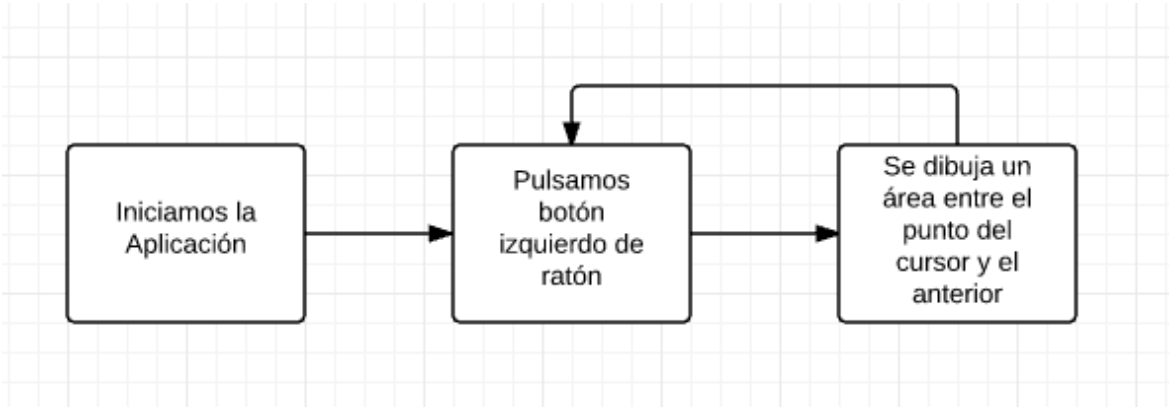


Figura 13: Funcionamiento básico de la Herramienta de Line Buffer 3D

5. Desarrollo

Con todo lo explicado anteriormente nos hemos podido hacer una idea general aunque bastante completa del funcionamiento de la aplicación, desde que se inicia la aplicación, que podemos elegir entre si manipular la cámara o utilizar las herramientas, hasta el uso particular de cada una de las herramientas. Una vez hablado de esto de manera general, pasamos a hablar más en detalle del funcionamiento interno de nuestra aplicación.

5.1 Desarrollo de osgATViewer

Empezaremos describiendo el mapa interno de la clase principal de la aplicación, osgATViewer. En ella definimos el primer elemento del grafo que hemos comentado al principio, y al que llamamos *root*, será un nodo de tipo Group, y vamos a ligarle varios nodos. Vamos a definir:

- El modelo que vamos a usar de ejemplo, que será un cuadrado.
- Los Shaders a utilizar en la aplicación.
- El manipulador de cámara de la clase UITrackballHandler.
- El manipulador del resto de eventos de la clase UIHandler.

Estos dos últimos, ya nombrados previamente, los asociamos al *root*. Además, vamos a asociar cada una de las herramientas a una tecla, como explicaremos más adelante y vamos a inutilizar la tecla ESC, para que no pueda utilizarse para salir de la aplicación, ya que esta tecla, como ya hemos comentado anteriormente, la utilizaremos en las herramientas de información y medida.

5.2 Desarrollo de UITrackballHandler

A continuación pasaremos a definir la clase UITrackballHandler, que como ya hemos dicho utilizaremos para manejar los eventos de manipulación de cámara. Esta clase va a heredar de la clase TrackballManipulator que nos provee OSG, y básicamente la vamos a utilizar de puente entre la aplicación y la clase padre ya nombrada. Aclaremos esto un poco: dispondremos de un campo que nos marcará si la manipulación de cámara esta activa o no, y si está activa delegaremos en la clase padre para manejar el evento de manipulación de cámara, y en caso contrario el evento lo manejará la clase UIHandler. El tener la manipulación de cámara activa o no se realizará pulsando la tecla “E”, como explicaremos más adelante, y que se

realizará con un método llamado setEnabled(bool enabled), que en caso de activarla se pone a *true*, y a *false* en caso contrario.

5.3 Desarrollo de UIHandler

Continuamos con la clase UIHandler, que como ya hemos dicho anteriormente heredará de la clase GUIEventHandler. En esta clase tenemos referencias del objeto de manipulación de cámara, el estado actual de la aplicación, navegación o uso de herramientas, un map para asociar cada herramienta a una tecla, y a la herramienta activa y la previa. Primero vamos a hablar sobre el estado de la aplicación: inicialmente empieza a 0, es decir, solo podremos manipular la cámara. En caso de querer manejar alguna herramienta, pulsaremos la tecla "E", y pasaremos a modo edición. Como hemos comentado previamente, desde la clase principal se asocian las herramientas a una tecla, en particular asociaremos la herramienta de Información a la tecla 1, la de Medidas a la 2, la de Buffer 3D a la 3, y finalmente la de Line Buffer 3D a la 4; por tanto, una vez el estado sea de Edición, podremos accionar cualquiera de estas cuatro teclas para activar una herramienta.

El manejo de estos eventos (pulsar las teclas nombradas), se realiza desde el método handle() de esta clase, en el que primero de todo vamos a comprobar si pulsamos la tecla "E", y si es así cambiamos de estado, sin importar nada de lo que haya hecho en el modelo. Se comprueba la tecla pulsada, y si es de cambio de estado, a que estado cambiamos, habilitando o deshabilitando la manipulación de cámara a conveniencia. Cuando el estado es de edición, se busca la herramienta asociada a la tecla que hayamos pulsado, y en caso de encontrarla, iniciamos la herramienta encontrada invocando su método begin(), la habilitamos, y ejecutamos el método finish() y deshabilitamos la herramienta usada anteriormente.

Una vez realizadas todas estas comprobaciones, delegamos en el método handle() de la herramienta activa, por tanto, podemos decir que esta clase también es una clase puente entre la aplicación y cada una de las herramientas. Más tarde explicaremos el funcionamiento concreto de los métodos handle(), begin() y finish() según la clase, ya que estos son métodos genéricos que invocamos desde la clase UIHandler y cada uno tendrá su definición.

5.4 Desarrollo de Herramientas

Pasamos ahora a definir el funcionamiento interno de cada una de las herramientas, pero para ello antes hemos de introducir la clase de la que heredarán todas ellas: PointCloudVizTools. Su definición es simple, disponemos de los métodos enableTool(bool value) para habilitar/deshabilitar la herramienta según el valor de "value", y los métodos begin(), finish() y handleToolEvents() que definimos en la clase hijo, como hemos comentado antes, y el método handle() que invoca al handleToolEvents() si la herramienta está activa.

5.5 Desarrollo de Information Tool

Dicho esto, podemos empezar definiendo la primera herramienta, la de Información. Disponemos de ciertas variables interesantes, como son _markerGroup que contendrá el conjunto de marcadores, o _state, que nos dirá el estado de la herramienta en cada momento, START, SHOWING_INFO o END_ SHOWING_INFO.

Empezaremos definiendo el método o función esencial en esta herramienta, al igual que en las otras, que será el handleToolEvents(). Cuando esta herramienta acaba de ser invocada, es decir, se ha pulsado la tecla 1, su estado es "START", y es entonces cuando removemos todos los marcadores que pueden haber sido previamente introducidos, y cambiamos el estado a "SHOWING_INFO", es decir, a partir de ahora la herramienta estará preparada para crear marcadores.

Con este segundo estado tenemos cuatro posibles eventos: pulsar el botón derecho del ratón, mover el cursor, pulsar la tecla ESC o la tecla de retroceso. El evento más sencillo de manejar es el último de ellos, en el que eliminaremos mediante la función removeChildren() el último marcador dibujado, solo en caso de que haya uno o más dibujados, en caso contrario el programa lanzaría un error. Otro caso es si pulsamos la tecla ESC, en cuyo caso eliminamos el marcador en curso, es decir, el ultimo que es el que vamos moviendo con el ratón, haciendo uso de la misma instrucción de antes, removeChildren(); y pasamos al estado "END_ SHOWING_INFO".

Este cambio de estado hará que solo surtan efecto dos eventos, volver a pulsar la tecla de ESC, en cuyo caso borraríamos todo y pasaríamos al estado "START", o borrar el último marcador dibujado con la tecla de retroceso, como hemos definido antes.

Definamos ahora el método move(), que será invocado cuando movamos el cursor. Vamos a utilizar una herramienta que nos provee OSG, los Polytope Intersector. Básicamente es la manera que tiene OSG de saber en qué punto está situado el cursor mediante unas intersecciones que explicaremos mejor en el anexo. Una vez calculado el punto del modelo en el que está situado el cursor, llamamos al método updateTextAndMarker() que explicaremos más adelante, aunque lo que haremos es sencillo, un marcador en el punto en el que está situado el cursor, y se moverá a medida que éste cambie de posición. Esto sucederá cada vez que movamos el cursor de lugar.

Por otro lado, el método push() será invocado cuando se pulse el botón izquierdo del ratón y aunque en esencia hará lo mismo que el move(), una vez se pulsa el botón el marcador que hasta ahora estábamos moviendo se dejará fijo en la posición en la que hayamos pulsado, y crearemos uno más que será el que irá acompañando al cursor a partir de ahora.

Hemos de definir por último el método finish() de esta clase: eliminaremos todos los marcadores con la instrucción _markerGroup->removeChildren(0, _markerGroup->getNumChildren()), es decir, desde el primer marcador hasta el último, sus textos, y pasaremos al estado "START".

5.6 Desarrollo de Measurement Tool

Pasamos a definir en detalle la herramienta de Medida. Dispondremos también de una variable _state que tendrá los estados START, DRAWINGLINE y ENDLINE; un array (osg:Vec3Array) en el que guardaremos los vértices que vayamos dibujando; y además tendremos una serie de variables que usaremos para mostrar las deltas y medidas de cada segmento, y la distancia total de todos los segmentos; más tarde los nombraremos.

Al crear el objeto de esta clase invocamos el método begin(), y en él crearemos un texto que será colocado en la parte inferior izquierda de la pantalla en el que mostraremos las distancias y deltas totales; esto será posible gracias al uso de la clase Camera de OSG. Esta herramienta estará afectada por cuatro eventos, bien sea mover el cursor del ratón, pulsar su botón izquierdo, una o dos veces, o pulsar ESC, este será manejado por el método handle().

La herramienta empieza con el estado "START", y una vez se pulsa el botón izquierdo del ratón, añadimos un vértice fijo en la posición donde hayamos pulsado, y otro que se moverá a medida que el cursor cambia de posición; la posición será

calculada al igual que antes, con los Polytope Intersector, y cambiaremos el _state a DRAWINGLINE, con lo que a partir de ahora el resto de eventos surtirán efecto.

Como acabamos de comentar, a partir de ahora, cada vez que movamos el cursor se moverá el último punto que hemos creado mediante la ejecución del método move(), de manera similar a como lo hacíamos en la herramienta de información, pero además aquí vamos a dibujar una línea entre los dos vértices que hemos añadido.

Otra funcionalidad que está disponible en esta herramienta, al igual que en la anterior, es la del uso de la tecla ESC: antes se borraba un marcador; y en este caso ejecutamos un método propio de la clase llamado quitLine(). En esencia, este método quita una arista, su distancia y sus deltas, y le restamos esta distancia a la distancia total; lo hace removiendo el último vértice de la variable _vertices y recalculando la distancia total y las deltas totales. Además, una vez pulsada esta tecla, el estado de la aplicación pasa a ser ENDLINE, y con este estado solo podremos pulsar una vez más la tecla ESC, lo que conllevará a la ejecución del método reset(): este método pondrá la aplicación a 0, eliminamos vértices, aristas y los textos, y recalculamos las distancias y las deltas totales para ponerlos a 0.

Por último habremos de definir el método finish(), que remueve la cámara que hemos comentado previamente con la que estamos visualizando el texto inferior izquierdo, y después invocamos el método reset() para remover todos los vértices, aristas y textos.

5.7 Desarrollo de Real Time Analysis Tool

A continuación describiremos la herramienta de Buffer 3D. En esta clase vamos a necesitar tener referencia del radio del círculo dibujado, que previamente va a ser 4 veces más pequeño que nuestro modelo, y que podremos modificarlo. Además tendremos que definir objetos Shader, para que la clase pueda saber la ubicación de los archivos donde estos están escritos, y una serie de Uniform:

- radiusUniform
- color1Uniform
- color2Uniform
- centerUniform

Éstas sirven de puente entre nuestra aplicación y las variables con las que el Shader podrá trabajar.

Como el resto de las herramientas, el objeto declarado de esta clase se inicializa con el método begin(), y en esta herramienta el comportamiento va a ser bastante básico: añadimos a la aplicación todos los Uniform y los dos archivos Shader previamente definidos, así solo en caso de seleccionar esta herramienta estaremos seguro de que se van a mostrar los resultados que muestra ésta. Como vamos a ver a continuación, esta herramienta maneja solo dos tipos de evento:

- Pulsar el botón izquierdo del ratón.
- Mover la rueda del ratón.

Pulsar el botón derecho del ratón nos va a permitir como siempre cambiar el punto en el que estamos dibujando, en este caso será el centro, y usaremos los métodos descritos anteriormente para obtener el punto de intersección. Además, informaremos al Shader de que hemos cambiado el centro desde donde éste habrá de dibujar, y lo haremos mediante la variable Uniform; aplicamos sobre dicho Uniform el método set(), pasándole como parámetro el valor que queremos que acepte el Shader.

Por último, tendremos que manejar los eventos de la rueda del ratón, en el que tendremos que distinguir el movimiento hacia arriba, que llamará al método doHigher(), y hacia abajo, que invocará al doLower(). El primero de ellos hará más grande el círculo, y lo hará de la siguiente manera: recoge la distancia recorrida por la rueda del ratón con el método getScrollingMotion(), incrementamos el radio actual _radius en 0.1* getScrollingMotion(), y actualizamos el Uniform de radio en esa misma proporción (de la misma manera que hemos comentado previamente), así el Shader podrá hacer cálculos correctamente.

El segundo método hará justo lo contrario, hará el círculo más pequeño, y en vez de incrementar disminuye el radio _radius.

Antes de finalizar, vamos a echar un rápido vistazo a la implementación del Shader, ya que entender como se ha diseñado es fundamental para entender la aplicación completamente. Su funcionalidad es algo sencilla, él será el encargado de pintar el círculo dibujado haciendo el degradado de color. Teniendo los Uniform como parámetro, calcula distancia entre los puntos, y en caso de ser menor que el radio, aplicamos una fórmula para pintar el fragmento que estamos analizando, y que al final del todo tengamos un degradado entre rojo al centro y azul en los bordes, aplicando la fórmula: color1*(1-dist/radius)+color2*dist/radius. En caso de ser mayor no pintamos nada.

5.8 Desarrollo de Real Time Analysis Tool 3D

Vamos a definir por último la herramienta de Line Buffer 3D. Ésta será en esencia muy parecida a la anterior: tendremos referencia de:

- El radio de los círculos dibujados, que previamente va a ser 4 veces más pequeño que nuestro modelo, y que podremos modificarlo.
- Los vértices almacenados en la variable _vertices que se modificará a medida que añadamos vértices pulsando el botón izquierdo del ratón.
- El número de vértices guardado en _numVertices.
- Una serie de objetos Shader, para que la clase pueda saber la ubicación de los archivos donde estos están escritos.
- Otra serie de Uniform, _radiusUniform, _color1Uniform, _ color2Uniform, _centerUniform, _vertices y _numVertices, con los que podremos proporcionar las variables con las que el Shader podrá trabajar.

Las acciones a realizar una vez la herramienta se activa y se invoca al método begin() son parecidas a las anteriores, y en definitiva lo que hacemos es añadir todas las variables Uniform a la aplicación, solo que en este caso añadiremos las dos nuevas también.

Los eventos a manejar de la herramienta anterior también aparecen en ésta, aunque podremos observar una diferencia respecto al evento de pulsar el botón izquierdo del ratón: a medida que vamos pulsando, vamos añadiendo vértices a la variable _vertices e incrementando la variable _numVertices, y se los proporcionamos al Shader para que realice los cálculos pertinentes, como explicaremos a continuación.

Además, en este caso habrá un evento más a manejar que la herramienta anterior; hablamos de cuando movemos el cursor: en este caso haremos como hacíamos en la herramienta de medidas, el último vértice disponible se desplazará a medida que se mueve el cursor del ratón, y esta información la vamos actualizando en el Shader para que el dibujo se vaya actualizando.

Los métodos doLower() y doHigher() serán iguales que en la herramienta de Buffer 3D, y la manera en que se invocan es también idéntica, usando la rueda del ratón o haciendo Scroll con el Touch Pad.

Por último, describiremos un poco lo que el Shader realiza internamente, así tendremos una descripción bastante completa de toda la aplicación. Si bien la anterior calculaba distancias entre el punto a pintar y el centro, aquí habrá que tener en cuenta que tenemos más de 1 vértice, por tanto se tendrá que ver que dicho punto

este a una distancia menor o igual a los segmentos formados entre cada par de puntos, y en caso afirmativo pintamos de la misma manera que antes, haciendo un degradado de color. Los cálculos empleados para saber si un punto está a la distancia menor o igual al radio respecto a estos segmentos son más avanzados, pero básicamente realizamos los cálculos por partes, si el punto a pintar está a la izquierda del segmento del que vamos a sacar la distancia (Respecto a la recta perpendicular que corta al segmento en el primer punto), miramos la distancia al primer vértice, en caso de estar al centro, hayamos la distancia al segmento, y sino al segundo vértice.

6. Resultados

Después de haber expuesto todos los detalles de cada una de las herramientas, damos paso a las imágenes, tanto en nuestra integración como en la herramienta PointCloudViz, con lo que podremos ver los resultados físicos de todo el trabajo realizado.

6.1 Imágenes de nuestra integración

Empezaremos con unas imágenes de la pantalla principal. Cabe decir que las el modelo que vamos a mostrar ahora es simplemente un ejemplo para ver el uso de las herramientas, éstas, una vez integradas se podrán usar con cualquier modelo cargado previamente, como hemos dicho anteriormente. En la Figura 14 podemos ver la imagen de la pantalla principal.

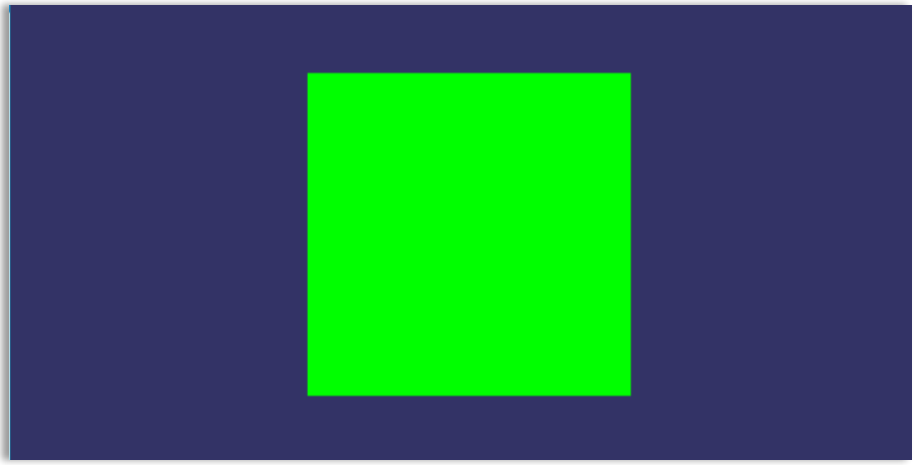


Figura 14: Pantalla principal de la aplicación

A continuación podremos, como ya hemos dicho anteriormente, realizar dos acciones, o bien movemos el modelo para situarlo de la manera que mejor podamos ver los detalles a analizar, o podemos activar las herramientas con la tecla E. Los resultados son visibles en la Figura 15.

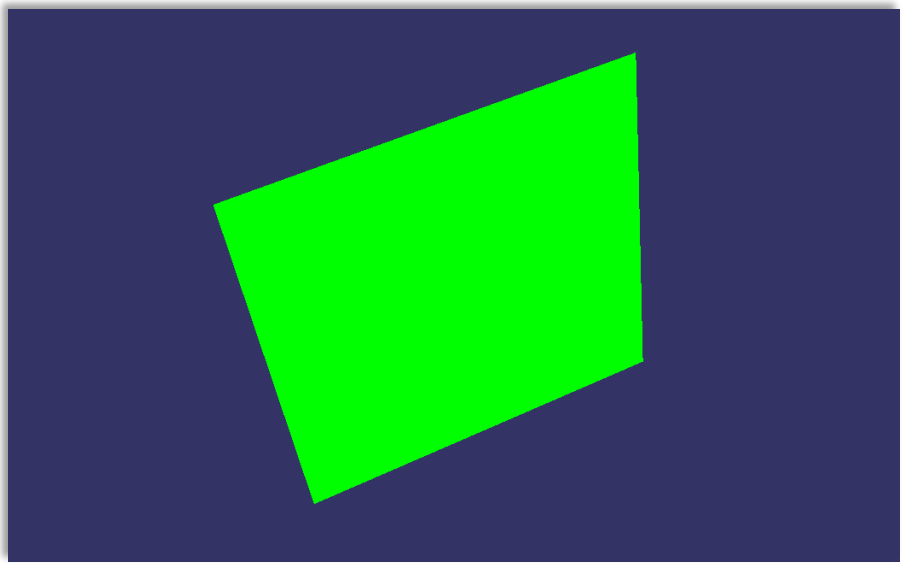


Figura 15: Modelo movido

Una vez pulsada la tecla E, cambiamos de estado, de manipulación de cámara a edición; podemos accionar cada herramienta con las teclas 1, 2, 3 ó 4. Observamos a continuación la Figura 16 con la primera herramienta.

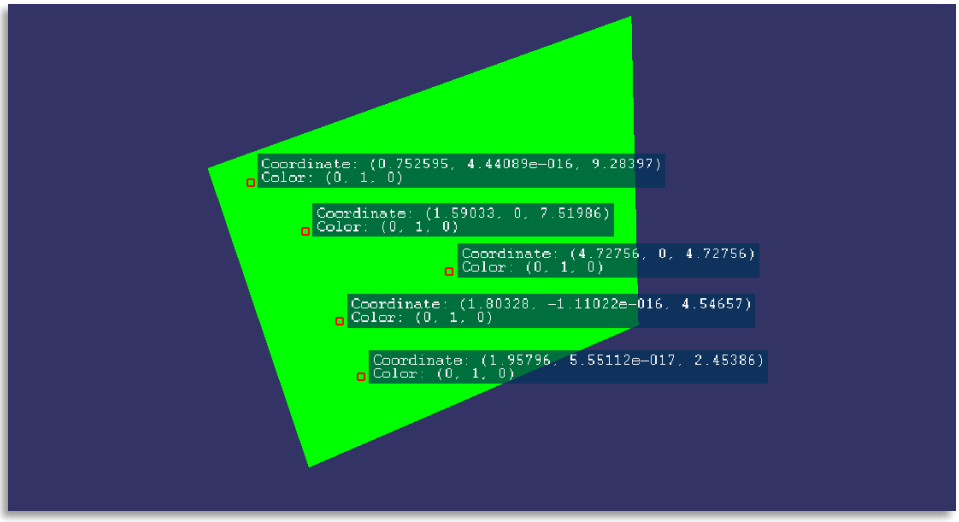


Figura 16: Primera herramienta

La segunda herramienta pasa por varias fases. Para empezar podemos ver una imagen de la herramienta en su uso corriente, a cada click se crea un nuevo segmento. El resultado es visible en la Figura 17.

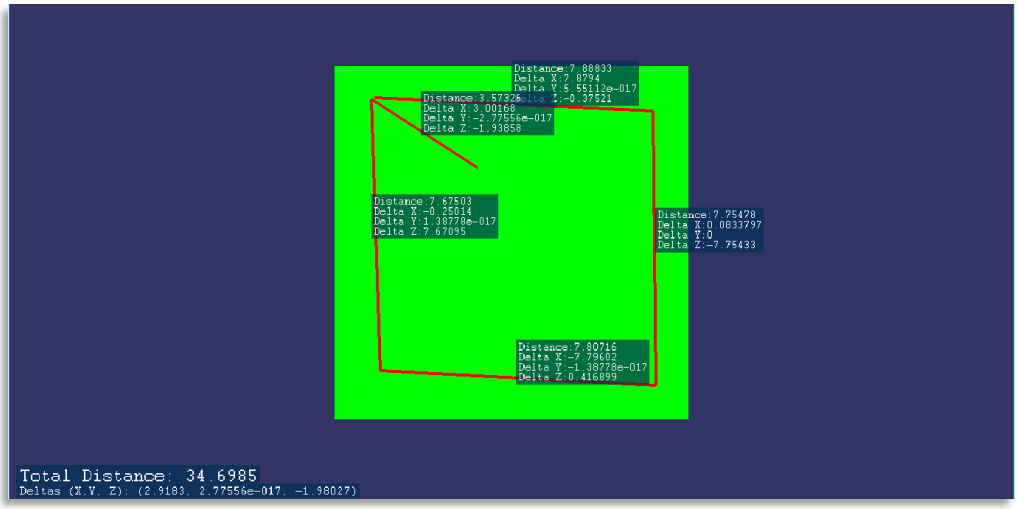


Figura 17: Segunda herramienta

Si se hace doble click, se finaliza la creación de segmentos, como podemos ver en la Figura 18, el último segmento tiene distancia 0.

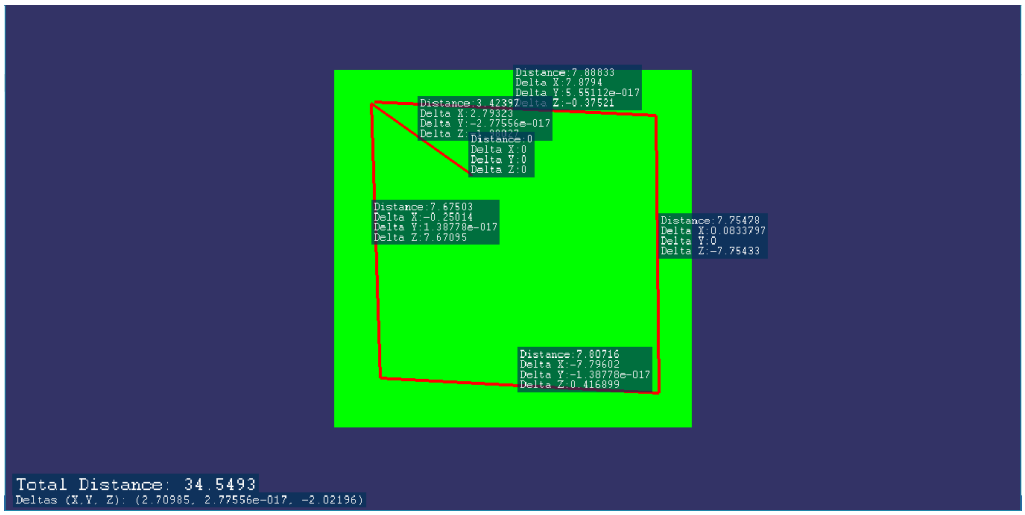


Figura 18: Segunda herramienta con doble click

Por último, si realizamos un conjunto de segmentos, y pulsamos la tecla ESC, se elimina el último segmento que estamos realizando. Veamos el resultado en la Figura 19.

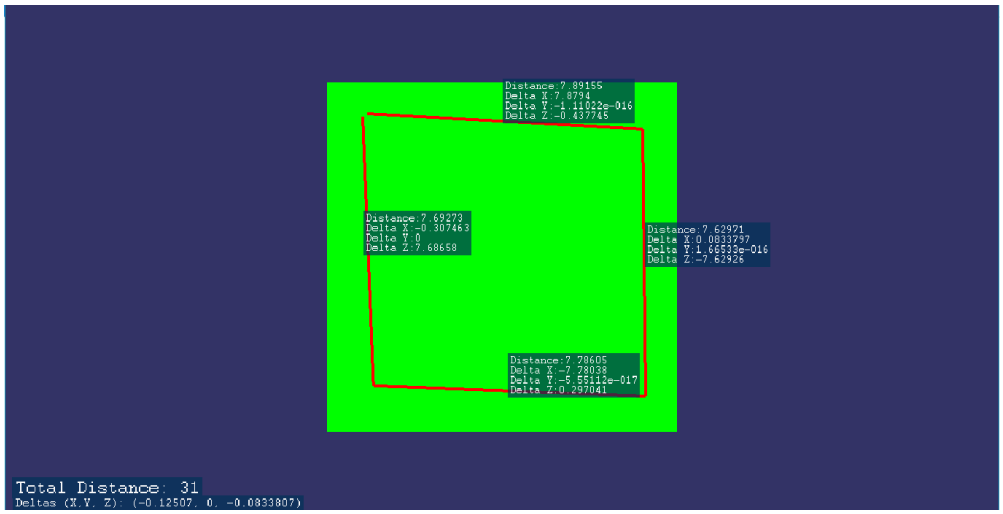
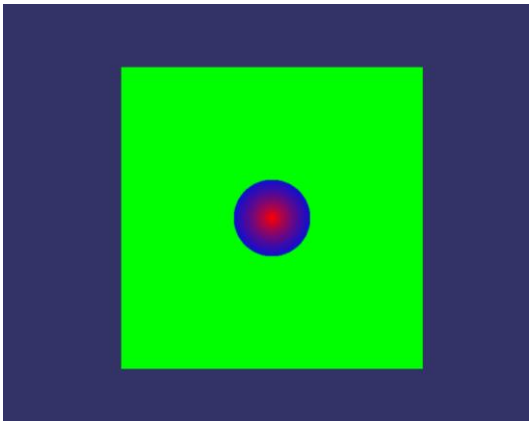
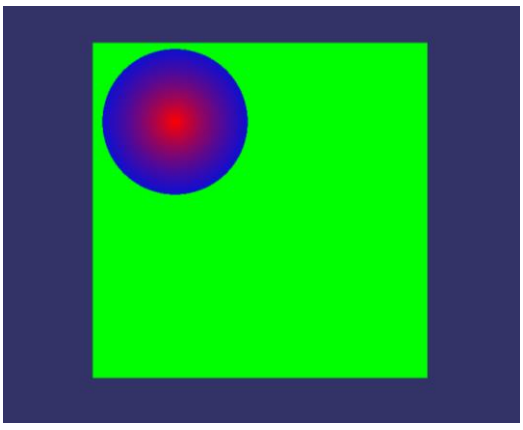


Figura 19: Segunda herramienta con tecla ESC

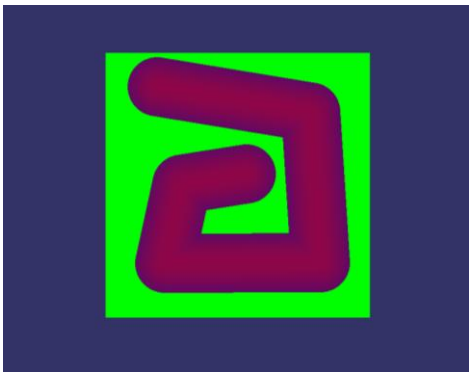
A continuación, en las Figuras 20 y 21 pasaremos a mostrar la 3ª herramienta. Inicialmente tendremos el área de influencia al medio, y podremos moverlo y modificar su tamaño:

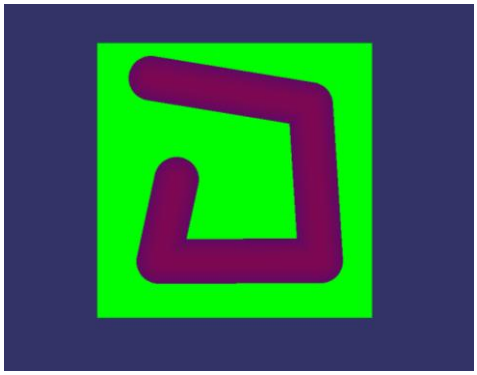




Figuras 20 y 21: Estado inicial y modificado de la Tercera herramienta

Por último mostraremos la 4ª herramienta. En su estado inicial no aparece nada, pero a medida que pulsamos con el ratón vamos creando segmentos como se muestra en la Figura 22, y que podemos hacer más grandes o pequeño a voluntad como vemos en la Figura 23.





Figuras 22 y 23: Cuarta herramienta con tamaño por defecto y disminuido

6.2 Imágenes de PointCloudViz

Como sabemos, este trabajo fue pensado para incluir una serie de herramientas en la aplicación PointCloudViz, además de ofrecer una API para poder integrarlas después en alguna otra aplicación similar. Podemos observar a continuación las Figuras 24, 25, 26 y 27 del resultado de integrar dichas herramientas.

Figura 24: Herramienta de información en PCV

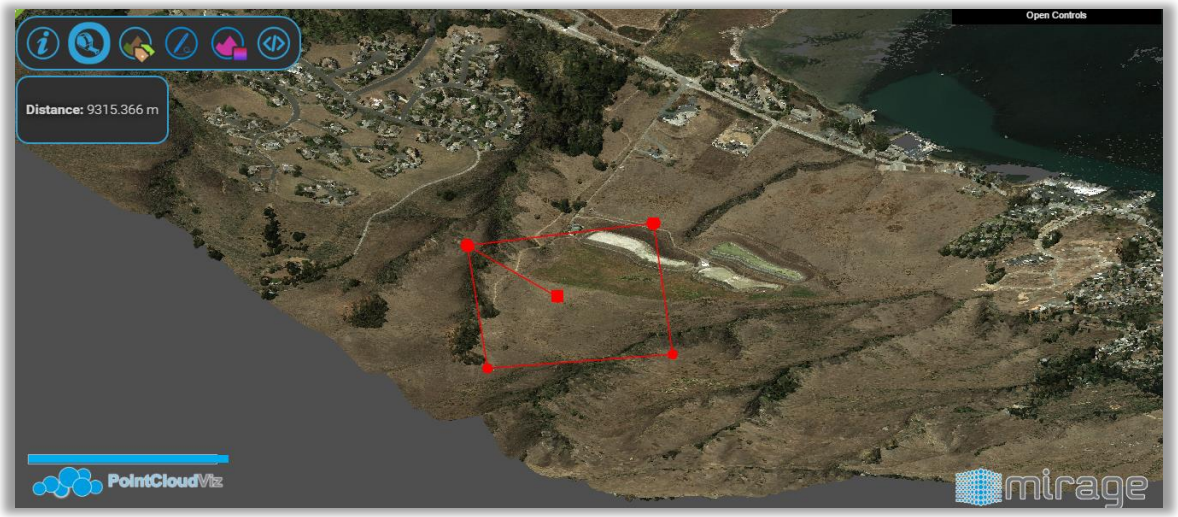


Figura 25: Herramienta de medidas en PCV

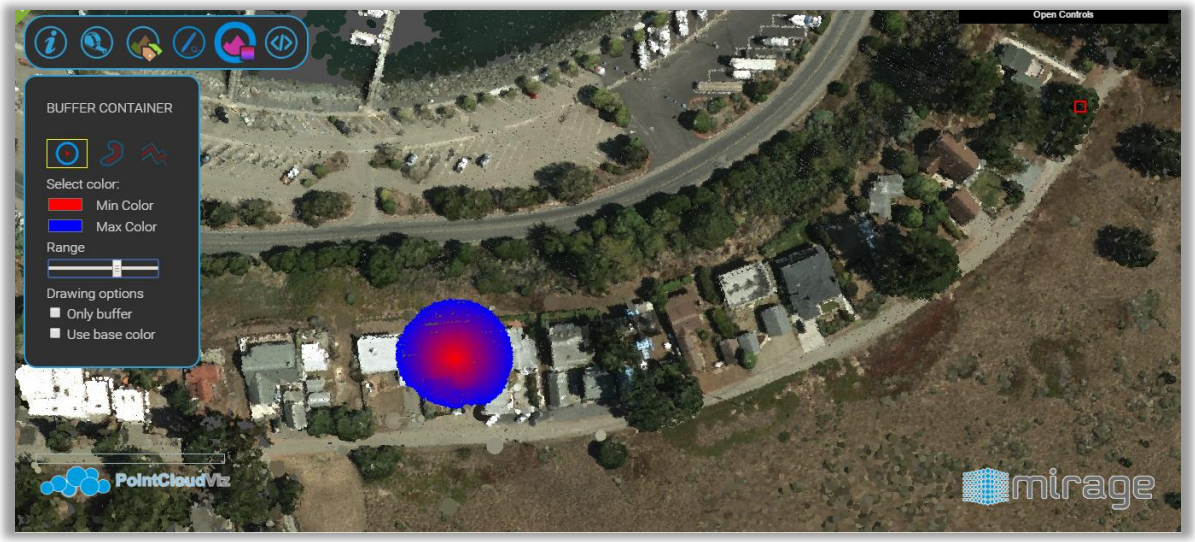


Figura 26: Herramienta de Buffer en PCV

Figura 27: Herramienta de LineBuffer en PCV

Estas dos últimas herramientas, además del funcionamiento que nosotros hemos creado, se les ha dado una funcionalidad extra: podemos visualizar únicamente lo que hayamos señalado con este buffer, tal y como vemos en la Figura 28 y la Figura 29.

Figura 28: Funcionalidad extra de LineBuffer en PCV

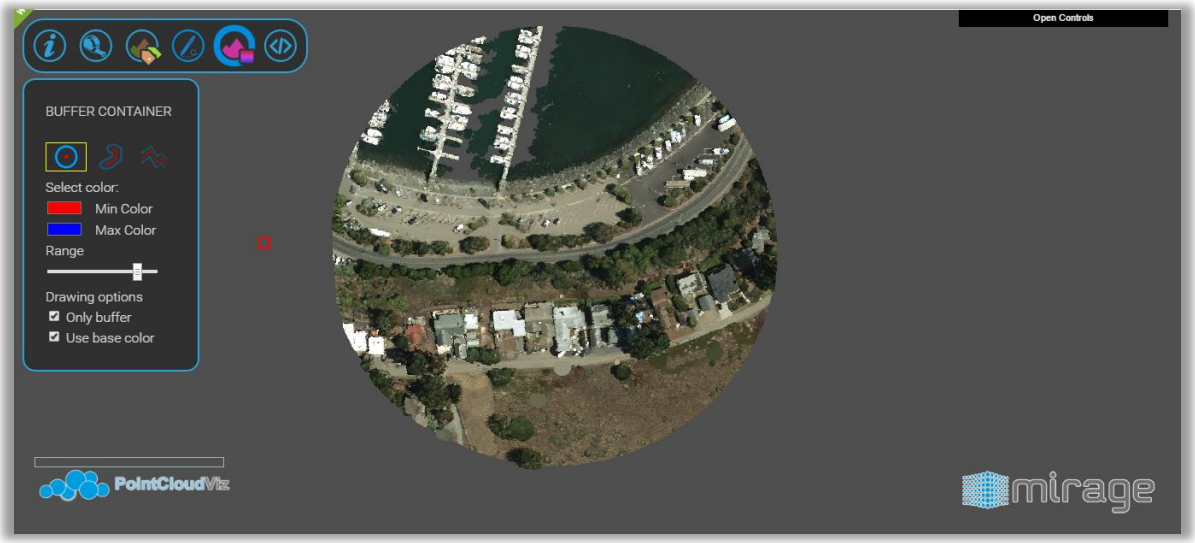


Figura 29: Funcionalidad extra de Buffer en PCV

7. Conclusiones

Bueno, llegados a este punto hemos de decir que la realización del trabajo ha sido satisfactoria. Como dijimos al principio en los objetivos, hemos cumplido todos y cada uno de ellos; hemos diseñado la herramienta de información, de medida, y los dos Buffer, tanto el puntual como lineal, y los hemos podido integrar satisfactoriamente en la aplicación PointCloudViz, como ya hemos visto en las imágenes. La elección de los precedentes fue la acertada, como detallamos a continuación:

- El uso de Shaders ha sido la parte más costosa, pero como hemos visto en los resultados, un poco de interés y conocimientos previos nos han ayudado a completar esta parte del trabajo.
- OSG ha resultado ser fácil de aprender a utilizar. Empecé con una guía de uso básico, y fui realizando poco a poco los ejercicios que se iban proponiendo, desde la creación de formas simples, hasta la creación de cámaras y la integración de los Shaders.
- El manejo de datos LiDAR no ha ido tan acompañado tanto de nuestra mano como la de Mirage Tech en su aplicación PointCloudViz, pero aun así, su comprensión ha sido bastante rápida y sencilla.
- Por último, el hecho de usar todos estos antecedentes ha sido clave para una integración más sencilla en PointCloudViz, ya que ésta hace uso de todos los antecedentes anteriores.

7.1 Trabajo futuro

Aunque este trabajo puede continuar desarrollándose poco a poco, hay ciertos aspectos que he planteado que podrían ser de utilidad para una futura modificación:

- Mostrar la medida real del diámetro de los Buffer, con lo cual podríamos saber el tamaño de un área de influencia, además de su posterior análisis.
- Unir ambos estados de la aplicación (Edición y Manipulación de Cámara) en uno, así no nos hubiera hecho falta una tecla para cambiar de estado, ni interrumpimos el uso de las herramientas. Las versiones de escritorio y web de PointCloudViz disponen de esta opción.
- Asociar efectos de sonido a cada una de las acciones, con lo que la relación entre usuario y aplicación se estrecha aún más.

- Capacidad de poder cargar modelos LiDAR desde nuestra aplicación, y que dicho funcionamiento pueda ser integrado posteriormente en otras aplicaciones. La versión de escritorio de PointCloudViz dispone de esta opción.
- Crear más herramientas de análisis, para analizar alturas máximas, tipo de terreno predominante (tierra, bosque...).

8. Bibliografía

[1] OSG. The OpenSceneGraph Project Website. [Internet]
Disponible en: http://www.openscenegraph.org/

[2] OSG. OSG Getting Started. [Libro]
Disponible en: www.packtpub.com/game-development/openscenegraph-30-beginners-guide

[3] OSG. Descarga de OSG. [Internet]
Disponible en: www.openscenegraph.org/index.php/download-section/stable-releases

[4] OSG Installation Guide. Get Started on OpenSceneGraph. [Internet]
Disponible en: www.openscenegraph.org/index.php/documentation/getting-started

[5] Shaders. Información General. [Internet]
Disponible en: https://es.wikipedia.org/wiki/Shader

[6] Shaders. Ejemplos de Shaders. [Internet]
Disponible en: https://www.shadertoy.com/

[7] FlightGear. Información General. [Internet]
Disponible en:
trac.openscenegraph.org/projects/osg//wiki/Sceenshots/Flightgear

[8] Capaware. Información General. [Internet]
Disponible en: http://www.capaware.org/

[9] LiDAR. Información General. [Internet]
Disponible en: https://es.wikipedia.org/wiki/LIDAR

[10] Nube de Puntos. Información General. [Internet]
Disponible en: https://es.wikipedia.org/wiki/Nube_de_puntos

[11] GLSL. Getting Started. [Internet]
Disponible en: http://joshbeam.com/articles/getting_started_with_glsl/

[12] GLSL. Documentation. [Internet]
Disponible en: https://www.opengl.org/documentation/glsl/

9. Glosario de Términos

PointCloudViz: **Aquí** tenemos su página Web. Nos proveen una versión Escritorio y una versión server.

Nube de puntos: Conjunto de puntos representado en un espacio de coordenadas. **Este** es un ejemplo.

API: Application Programming Interface, es decir, interfaz de programación de aplicaciones. Es un conjunto de funciones que vamos a dar a conocer al posible usuario para integrar estos funcionamientos en su propia aplicación.

GLSL: OpenGL Shading Language, también conocido como **GLslang**, una tecnología parte del API estándar **OpenGL**, que permite especificar segmentos de programas gráficos (Shaders) que serán ejecutados sobre el **GPU**. Su contrapartida en DirectX es el **HLSL**.

Delta: es la diferencia en (x,y,z) entre dos puntos, y cuyo modulo es la distancia entre ambos puntos.

OSG: Open Scene Graph, es un motor gráfico que está escrito en el lenguaje de programación C++ y que suele ser empleado en el desarrollo de aplicaciones, como simulación visual, videojuegos, realidad virtual... Se usan los grafos para construir una escena completa, con dependencias entre los nodos que están relacionados entre sí.

Polytope Intersector: Es una herramienta provista por OSG que nos permite calcular puntos de intersección mediante uso de lo que llaman Politopos. Básicamente se proyecta un prisma sobre un el plano, y hacemos un recorrido hasta encontrar el punto en el que estamos interesados.

Uniform: es un tipo de variable usada en los Shaders, y que servirá de puente entre la aplicación y el código de Shader, es decir, su valor será definido fuera, se lo proporcionamos al Shader, y éste puede modificar el valor a voluntad.

Motor gráfico: es un conjunto de librerías que se pueden utilizar para el diseño de elementos gráficos, así como su funcionamiento; son utilizados comúnmente para diseñar videojuegos.

10. Anexo

Se proporciona un link a la aplicación almacenada en Dropbox, para complementar toda la información aquí escrita con el proyecto, que además contiene el código fuente:
https://www.dropbox.com/sh/fduowaq7r0052ps/AADIaOlwAkbjvl3olwXcgXx2a?dl=0