Universidad Nacional de Loja

Facultad de la Energía, las Industrias y los Recursos Naturales no Renovables

Carrera de Ingeniería en Electrónica y Telecomunicaciones

Estudio e implementación de simulador de mapa topográfico interactivo para la facultad de energía de la UNL

Trabajo de Titulación previo a optar por el Título de Ingeniero en Electrónica y Telecomunicaciones

AUTOR:

Carlos Fabricio Condolo Castillo.

DIRECTOR:

Ing. Rodolfo Pabel Merino Vivanco, Ms.Sc.

LOJA-ECUADOR

2022

Certificación

Ing. Rodolfo Pabel Merino Vivanco Mg.Sc.

Director del trabajo de titulación.

CERTIFICA:

Haber dirigido, asesorado, revisado y corregido el presente trabajo de titulación, en su proceso de investigación cuyo tema versa en **"Estudio e implementación de simulador de mapa topográfico interactivo para la facultad de energía de la UNL"** previo a la obtención del título de ingeniería en **Electrónica y Telecomunicaciones,** realizado por el señor egresado: **Carlos Fabricio Condolo Castillo** mismo que cumple con la reglamentación y políticas de investigación, por lo que autorizo su presentación para su posterior sustentación y defensa.

Loja, 1 de febrero del 2022

Firmado electrónicamente por:
RODOLFO PABEL
MERINO VIVANCO

Ing. Rodolfo Pabel Merino Vivanco Mg.Sc

DIRECTOR DEL TRABAJO DE TITULACIÓN

Autoría

Yo **CARLOS FABRICIO CONDOLO CASTILLO**, declaro ser el autor del presente trabajo de titulación y eximo expresamente a la Universidad Nacional de Loja y a sus representantes jurídicos de posibles reclamos o acciones legales por el contenido de la misma.

Adicionalmente acepto y autorizo a la Universidad Nacional de Loja, la publicación de mi trabajo de titulación en el Repositorio Institucional – Biblioteca Virtual.

Firma:

Cédula de identidad : 1105267734

Fecha: Loja, 15 de Julio del 2022

Correo electrónico: carlos.condolo@unl.edu.ec

Celular: 0982906119

Carta de autorización del trabajo de titulación por parte del autor, para la consulta, reproducción parcial o total y publicación electrónica del texto completo.

Yo **Carlos Fabricio Condolo Castillo,** declaro ser autor del trabajo de titulación denominado: **"Estudio e implementación de simulador de mapa topográfico interactivo para la Facultad de Energía de la UNL",** como requisito para optar al grado de: **Ingeniero en Electrónica y Telecomunicaciones;** autorizo al Sistema Bibliotecario de la Universidad Nacional de Loja para que con fines académicos, muestre al mundo la producción intelectual de la Universidad, a través de la visibilidad de su contenido de la siguiente manera en el Repositorio Institucional:

Los usuarios pueden consultar el contenido de este trabajo en el repositorio Institucional, en las redes de información del país y del exterior, con las cuales tenga convenio la Universidad.

La Universidad Nacional de Loja, no se responsabiliza por el plagio o copia del trabajo de titulación que realice un tercero.

Para constancia de esta autorización, en la ciudad de Loja, a los quince días del mes de julio del dos mil veintidós.

Firma:

Autor: Carlos Fabricio Condolo Castillo **Cédula:** 1105267734

Dirección: Loja, Las Peñas

Correo Electrónico: carlos.condolo@unl.edu.ec **Celular:** 0982906119

DATOS COMPLEMENTARIOS:

Director de Trabajo de Titulación: Rodolfo Pabel Merino Vivanco, Mg.Sc.

Dedicatoria

Dedico este proyecto de trabajo de titulación a Dios, por bendecirme en cada etapa de mi vida, dándome fortaleza para sobrellevar las adversidades.

A mis padres por haberme apoyado incondicionalmente a lo largo de mi vida, que con su amor y cariño han inculcado en mí el espíritu de superación, valores y principios que me han sido parte primordial para cumplir todas mis metas.

A mis hermanos que han sido un claro ejemplo de superación y coraje, siempre han estado a mi lado alentándome, por su cariño y apoyo absoluto.

A mi familia y amigos que a través de sus consejos y enseñanzas me permitieron ser un mejor ser humano cada día.

A mi director de trabajo de titulación quien con su paciencia, compresión y apoyo me compartió sus conocimientos para poder culminar este proyecto.

Carlos Fabricio Condolo Castillo

Agradecimiento

Mi eterno agradecimiento a la UNIVERSIDAD NACIONAL DE LOJA, a las Autoridades y Docentes que forman parte de la Facultad de la Energía, Las Industrias y Los Recursos Natrales No Renovables, especialmente a los Docentes de la Carrera de Ingeniería en Electrónica y Telecomunicaciones que han inculcado en mi sus conocimientos y valores a lo largo de mis años de estudio.

A si mismo al Ing. Rodolfo Pabel Merino Vivanco director de mi trabajo de titulación que a lo largo del desarrollo del proyecto siempre me apoyado de manera desinteresada, orientándome con sus capacidades y conocimientos en el progreso de este trabajo de investigación.

Al Ing. Jorge Isaac Peralta que de manera muy amable me apoyo en el proceso de desarrollo del proyecto, en general agradezco a todas las personas que de una u otra manera me apoyaron para poder culminar mi trabajo de titulación.

Carlos Fabricio Condolo Castillo

Índice de Contenidos

Portada ..... **i**

Certificación ..... **ii**

Autoría ..... **iii**

Carta de autorización ..... **iv**

Dedicatoria ..... **v**

Agradecimiento ..... **vi**

Índice de Contenidos ..... **vii**

Índice de Tablas xi

Índice de Figuras xi

Índice de Anexos xiv

1. Titulo **1**

2. Resumen **2**

2.1. Abstract 3

3. Introducción **4**

4. Marco Teórico **6**

4.1. Historia de la Realidad Aumentada 6

4.2. Realidad Aumentada Y La Educación 9

4.3. Realidad Aumentada 10

4.4. Tipos de realidad aumentada 11

4.4.1. Realidad aumentada geolocalizada. 11

4.4.2. Realidad Aumentada Basada en Marcadores. 12

4.4.3. Realidad aumentada basada en reconocimiento de imágenes. 13

4.5. Niveles de realidad aumentada. 13

4.5.1. Nivel 0 (enlazados con el mundo físico). 13

4.5.2. Nivel 1 (AR con marcadores). 14

4.5.3. Nivel 2 (AR sin marcadores). 14

4.5.4. Nivel 3 (Visión aumentada). 15
4.6. Proceso de la realidad aumentada 16
4.6.1. Captación de la escena. 17
4.6.2. Identificación de la escena o el entorno. 18
4.6.3. Aumento de la realidad. 18
4.6.4. Visualización. 18
4.7. Agentes relevantes en la provisión de servicios de realidad aumentada 19
4.7.1. Fabricantes de Dispositivos. 19
4.7.2. Desarrolladores del software de realidad aumentada. 19
4.7.3. Mundo digital y digitalización. 19
4.7.4. Red: operadores de telecomunicaciones. 20
4.8. Mapa topográfico 20
4.8.1. Curvas de nivel. 21
4.9. Modelos Geográficos 22
4.10. Modelos Hidrológicos 23
5. Materiales y Métodos 23
5.1. Materiales 24
5.1.1. Materiales de software 24
5.1.1.1. Paquete ARSandbox 24
5.1.1.2. Linux 24
5.1.1.3. Solidworks 25
5.1.2. Materiales de Hardware 25
5.1.2.1. Computadora 25
5.1.2.2. Cámara Microsoft Kinect 3D y Adaptador USB a 2.0 26
5.1.2.3. Proyector de video digital 27
5.1.2.4. Caja de arena 27
5.1.2.5. Estructura de aluminio 28
5.1.2.6. Arena 28
5.1.3. Presupuesto 29

5.2. Métodos 30
5.2.1. Método Incremental 30
5.2.2. Método inductivo 30
5.2.3. Método deductivo 30
5.2.4. Método analítico 30
5.3. Especificaciones de la metodología 30
5.3.1. Fase conceptual 30
5.3.2. Construcción de la estructura 31
5.3.3. Calibración del prototipo 31
5.4. Descripción General 31
5.4.1. Configuración inicial 31
5.4.2. Determinación de la ecuación base 34
5.4.3. Asignación de colores 35
5.4.4. Calibración 37
5.5. Selección de sistema operativo 39
5.5.1. Proceso de instalación 39
5.5.2. Precio 40
5.5.3. Posibilidad de mejoramiento 41
6. Resultados 41
6.1. Instalación de los recursos de software 41
6.1.1. Instalar Linux 42
6.1.2. Instalar controladores de tarjeta gráfica 42
6.1.3. Instalar el kit de herramientas de desarrollo de Vrui VR 43
6.1.4. Instalar el paquete de video Kinect 3D 46
6.1.5. Instalar el Sandbox de realidad aumentada 47
6.1.6. Conectar y configurar la cámara 3D 49
6.1.7. Alinear Kinect sobre la zona de pruebas 49

6.1.8. Medir la ecuación de plano base de Sandbox 50
6.1.9. Medir las posiciones de las esquinas de la caja 3D de Sandbox 54
6.1.10. Proyección de posición 60
6.1.11. Calibrar AR Sandbox 62
6.1.12. Ejecución AR Sandbox 64
6.1.13. Crear icono en el escritorio para abrir AR Sandbox 65
6.2. Diseño del simulador de mapa topográfico interactivo 69
6.2.1. Modelo base a seguir 69
6.2.2. Prototipo 70
6.3. Construcción de la estructura física de la caja de arena 73
6.4. Pruebas de Funcionamiento 78
6.5. Cálculo de porcentaje de diferencia 81
7. Discusión 85
8. Conclusiones 86
9. Recomendaciones 87
10. Bibliografía 88
11. Anexos 90

Índice de Tablas

Tabla 1: Costos de materiales e insumos 29
Tabla 2: Especificaciones técnicas kinect 90
Tabla 3: Especificaciones técnicas proyector de video digital 91
Tabla 4: Especificaciones técnicas computadora 92

Índice de Figuras

Figura 1: Sistema de realidad aumentada de Sutherland 6
Figura 2: ARToolkit 7
Figura 3: H-Pacman 7
Figura 4: Wikitude 8
Figura 5: Zapatillas Adidas Originales, Augmented Reality 8
Figura 6: Código QR 13
Figura 7: Marcador de realidad aumentada 14
Figura 8: Realidad aumentada con geolocalización 15
Figura 9: Realidad aumentada con Google Glass 16
Figura 10: Lentes biónicas de AR 16
Figura 11: Fases del proceso de RA 17
Figura 12: Representación de equidistancia. 21
Figura 13: Representación del concepto de curvas de nivel 22
Figura 14: computadora portátil 26
Figura 15:Kinect y adaptador 27
Figura 16: proyector de video digital 27
Figura 17: Caja de arena 28
Figura 18: Estructura de aluminio 28
Figura 19: Diagrama de funcionamiento 31
Figura 20: Método de luz estructurada 33
Figura 21: Efecto del parámetro alfa 35
Figura 22: Elemento de Calibración 37
Figura 23: Análisis Comparativo 39
Figura 24: Archivo Ejecutable 39
Figura 25: Terminal de linux 40
Figura 26: Licencia de uso 40

Figura 27: Sitio web del desarollador 41
Figura 28: Verificación de controladores de tarjeta gráfica..... 43
Figura 29: Instalación de kit de herramientas de Vrui VR 43
Figura 30: Instalación de kit de herramientas de Vrui VR 44
Figura 31: Ventana con globo terráqueo giratorio..... 44
Figura 32: Menú principal de la Ventana con globo terráqueo giratorio..... 45
Figura 33: Instalación de paquete de video Kinect 3D..... 46
Figura 34: Instalación de paquete de video Kinect 3D..... 46
Figura 35: Instalación de paquete de video Kinect 3D..... 47
Figura 36: Comprobación de la lista impresa 47
Figura 37: Instalación de Sandbox de realidad aumentada..... 48
Figura 38: Instalación de Sandbox de realidad aumentada..... 48
Figura 39: Comprobación de la lista impresa 49
Figura 40: Alineación de Kinect sobre la zona de pruebas..... 49
Figura 41: Comando para medir ecuación de plano base 50
Figura 42: Lista de opciones de configuración de Kinect..... 50
Figura 43: lista de opciones para extraer planos..... 51
Figura 44: ajuste de la vista de profundidad 51
Figura 45: Código de respuesta a la extracción del plano 52
Figura 46: Comando para abrir archivo BoxLayot.txt..... 52
Figura 47: Ventana de archivo BoxLayout.txt..... 53
Figura 48: Ventana de archivo BoxLayout.txt..... 53
Figura 49: ventana de archivo BoxLayout.txt..... 54
Figura 50: Código para medir la posición de las esquinas de la caja 54
Figura 51: Lista de opciones para la configuración del Kinect 55
Figura 52: lista de opciones para medir posiciones 3D 55
Figura 53: selección de la esquina inferior izquierda de la caja 56
Figura 54: selección de la esquina inferior derecha de la caja..... 56
Figura 55: selección de la esquina superior izquierda de la caja 57
Figura 56: selección de la esquina superior derecha de la caja..... 57
Figura 57: Código del sistema para medir la posición de las extensiones 3D..... 58
Figura 58: código del sistema para medir la posición de las extensiones 3D..... 58
Figura 59: Comando para editar archivo BoxLayout.txt 59
Figura 60: Ventana de archivo BoxLayout.txt..... 59

Figura 61: Ventana de archivo BoxLayout.txt 60
Figura 62: Ventana de archivo BoxLayout.txt 60
Figura 63: Comando para ajuste del área de la caja de arena 61
Figura 64: Proyección de la ventana de ajuste Xbachground 61
Figura 65: Objeto de Alineación 62
Figura 66: Comando para el ajuste del ancho y el alto de la imagen 63
Figura 67: Captura de punto de unión 63
Figura 68: Captura de puntos de unión 64
Figura 69: Proyección de relieve 65
Figura 70: Comando para crear el script 65
Figura 71: ventana del archivo script 66
Figura 72: Comando para ejecutar el script 67
Figura 73: Comando para crear el icono en el escritorio 67
Figura 74: Archivo de texto para crear el icono 68
Figura 75: Modificación del nombre de usuario 68
Figura 76: Comando para hacer el archivo ejecutable 69
Figura 77: Icono del AR Sandbox 69
Figura 78: modelo base a seguir 70
Figura 79: vista lateral izquierda de la caja de arena en Solidworks 71
Figura 80: Vista lateral de la caja de arena en Solidworks 71
Figura 81: Vista frontal de la caja de arena en Solidworks 72
Figura 82: vista lateral derecha de la caja de arena en Solidworks 72
Figura 83: vista lateral izquierda de la caja de arena en Solidworks 73
Figura 84: soporte de caja de arena 73
Figura 85: Acoplamiento de ruedas a la caja de arena 74
Figura 86: Estructura terminada de la caja de arena 74
Figura 87: Estructura de Aluminio 75
Figura 88: Conector de plástico para perfiles de aluminio 75
Figura 89: Estructura completa de la caja de arena 76
Figura 90: Limpieza de la arena 76
Figura 91: Encuadre de la proyección dentro de la superficie 77
Figura 92: Estructura para acoplamiento 78
Figura 93: proyección Topográfica 78
Figura 94: Modificación de la superficie para generar relieves 79

Figura 95: relieve de superficie montañosa 80
Figura 96: Representación de colores del relieve 80
Figura 97: Representación de elevaciones 81
Figura 98: Puntos de análisis 82
Figura 99: Valor del punto A 83
Figura 100: Ubicación del punto B 83
Figura 101: Valor del punto B 84
Figura 102: Pendiente con pendiente pronunciada (acantilado) 85
Figura 103: Kinect y adaptador 90
Figura 104: proyector de video digital 91
Figura 105: computadora portátil 92

Índice de Anexos

Anexo 1: Especificaciones Técnicas 90
Anexo 2: Manual de operación y mantenimiento 93
Anexo 3: Código en C++ del programa 96
Anexo 4: Certificado de Traducción 134

1. Titulo

“Estudio e implementación de simulador de mapa topográfico interactivo para la facultad de energía de la UNL”

2. Resumen

En la presente investigación se desarrolló la revisión bibliográfica de los sistemas de simulación en 3D interactivos para topografía y la construcción de un dispositivo de realidad aumentada que permita crear modelos topográficos de una superficie moldeable para cubrir con las necesidades académicas de la carrera de Minas de la facultad de Energía de la Universidad Nacional de Loja. De la misma forma se realizaron manuales que le permitan al usuario tener un uso y mantenimiento adecuado del simulador para que puedan disfrutar de todas sus funciones.

El proyecto se inicia con el estado del arte de los conocimientos de realidad aumentada y de los requerimientos de software y de hardware que se necesitan para llevar a cabo la construcción de la caja de arena interactiva los cuales son de libre difusión (Open-Source). Esta implementación está constituida por dos etapas principales, la primera etapa está formada por la construcción de la estructura de la caja de arena; en la segunda etapa se realizó el ajuste y calibración del software de realidad aumentada, el mismo que proyectará la imagen del relieve sobre la superficie de arena.

Con todo esto se pretende crear nuevos medios de enseñanza que le permita al docente impartir sus conocimientos a los estudiantes de una forma más interactiva y práctica en materias como petrografía, cartografía, sistemas de información geográfica, aplicación de softwares mineros, geotecnia minera, etc.

Palabras clave: Simulador 3D, Realidad Aumentada, Curvas de nivel, Mapa topográfico, Aprendizaje interactivo.

2.1. Abstract

In the present investigation, the bibliographic review of a 3D interactive simulation systems for topography and the construction of an augmented reality device that allows the creation of topographic models of a moldable surface to cover the academic needs of the Mining career of the National University of Loja. In the same way, user manuals were made that allow to the user to have an adequate use and maintenance of the simulator in order that they can take advantaje of all its functions.

The project begins with the state of the art of augmented reality knowledge and the software and hardware requirements needed to carry out the implementattion of the interactive sandbox, which are freely available (Open-Source). . This implementation is constituted by two main stages, the first one is formed by the construction of the structure of the sandbox; and in the second stage, the adjustment and calibration of the augmented reality software was carried out, the same one that will project the image of the relief on the sand surface.

With all of this, the idea is to create new means of teaching that allow teachers to impart their knowledge to the students in a more interactive and practical way in subjects such as petrography, cartography, geographic information systems, application of mining software, mining geotechnics, etc.

Keywords: 3D simulator, Augmented Reality, Contour lines, Topographic map, Interactive learning.

3. Introducción

Mucho se ha hablado de las formas de enseñanza que existen en la actualidad y de la manera en cómo pueden afectar al correcto aprendizaje de los estudiantes, sin embargo la desigualdad que existe para poder acceder a la educación digital de calidad y la infraestructura tecnología en nuestro país ha ido disminuyendo a lo largo de estos años, pero a pesar de esto se sigue teniendo la dificultad para introducirse a estos medios tecnológicos, es por ello que a nivel mundial se trabaja diariamente en el desarrollo de tecnologías que mejoren la metodología de enseñanza.

La realidad aumentada (AR) a lo largo de los años se ha convertido en una herramienta tecnológica que ha incidido en el desarrollo de actividades cotidianas ya sea de forma directa e indirecta, donde la educación ha tenido grandes avances, hablando de métodos y formas de impartir los conocimientos a nivel de formación superior. Esta tecnología mejor conocida como AR es un conjunto de elementos y dispositivos que permiten agregar información virtual a la información física existente.

La potenciación de los espacios educativos de la Universidad Nacional de Loja (UNL) se ha convertido en uno de los ejes más importantes teniendo en cuenta que los mismos les permiten a los estudiantes desarrollar sus destrezas y criterios de una forma más eficiente.

El estudio se desarrolló con la finalidad de solventar la necesidad de la carrera de Minas de la Facultad de Energía de la UNL, la cual está centrada en crear nuevas metodologías de enseñanza basadas en tecnologías que le permiten al estudiante absorber los conocimientos impartidos teóricamente en clases de una forma más dinámica, esto con el fin de reforzar dichos conocimientos y se pueda alcanzar una comprensión total.

El prototipo de simulador de mapa topográfico interactivo permite combinar y moldear un entorno físico del mundo real (superficie de arena) con elementos virtuales (proyección de mapa topográfico sobre la superficie) los cuales se van acoplando con la superficie moldeada con anterioridad.

A partir de una revisión bibliográfica de los sistemas de simulación en 3D interactivos para topografía se analizará las diferentes alternativas de software y los conceptos generales de realidad aumentada, posteriormente se desarrolló el diseño de la caja de realidad aumentada y se detallará a brevedad los materiales y recursos necesarios para la construcción del prototipo.

Por último, se realizaron los manuales de uso y mantenimiento del simulador para minimizar el uso inadecuado por parte de los usuarios.

Para el desarrollo óptimo del proyecto se plantearon previamente los siguientes objetivos:

Objetivo General

- Implementar un simulador de mapa topográfico interactivo que permitan a los docentes de la Facultad de Energía de la UNL mejorar los métodos educativos.

Objetivos Específicos

- Realizar una revisión bibliográfica de los sistemas de simulación en 3D interactivos para topografía.
- Identificar los requerimientos y necesidades de los docentes para el mejoramiento del aprendizaje.
- Diseño y réplica de un simulador de mapa topográfico interactivo que cubra las necesidades académicas requeridas para la docencia en la Facultad de Energía de la UNL.

4. Marco Teórico

4.1. Historia de la Realidad Aumentada

El primer instrumento que guarda relación con la Realidad Aumentada (AR) apareció en la segunda mitad del siglo XX, concretamente en el año 1962 de la mano de Morton Heilig. Se trataba de un prototipo denominado Sensograma, que junto con 5 filmes cortos permitía aumentar la experiencia del espectador a través de los sentidos de la vista, el olfato, el tacto y el oído(Castillo Otegui, 2017).

El primer sistema de realidad aumentada fue creado por Ivan Sutherland en 1968 y utiliza un casco visual capaz de representar objetos 3D simples renderizados en tiempo real. Se utilizaron dos sistemas de seguimiento para calcular las grabaciones de la cámara, uno mecánico y el otro basado en ultrasonido (Figura 1).

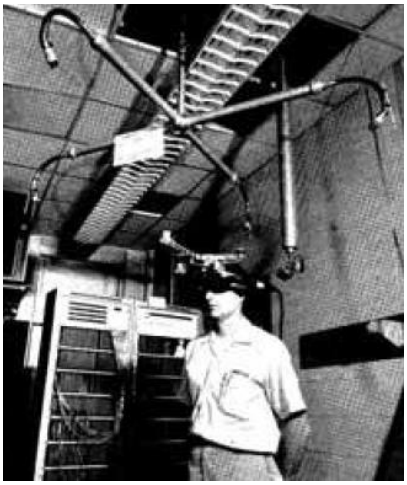

Figura 1: Sistema de realidad aumentada de Sutherland
Fuente: (Gonzáles Morcillos, Carlos; Fernández vallejo et al., 2012)

"En 1975, Myron Krueger, un artista informático estadounidense desarrolló la primera interfaz de realidad virtual, que permitía a sus usuarios manipular e interactuar con objetos virtuales y hacerlo en tiempo real"(Pesce, 2020).

El primer sistema de AR funcional fue desarrollado por Louis Rosenberg en 1992 en los Laboratorios Armstrong de la USAF (United States Air Force), es cual fue un sistema robótico altamente complejo llamado virtual Fixtures diseñado para compensar la falta de potencia de procesamiento de gráficos 3D de alta velocidad. Permitió la productividad humana habilitando superposiciones sensoriales en el espacio de trabajo.

En 1997, investigadores de la Universidad de Columbia presentaron The Touring Machine, el primer sistema de realidad aumentada móvil del mundo (MARS). Utiliza un sistema de visión en perspectiva que combina directamente gráficos 2D y 3D proyectados en una pantalla transparente con imágenes reales.

En 1999, se produjo un gran avance en el desarrollo de la realidad aumentada cuando Hirokazu Kato creó ARToolkit, una biblioteca de potentes herramientas para crear aplicaciones de realidad aumentada (figura 2). ARToolkit ha hecho que la realidad aumentada sea accesible para más investigadores y desarrolladores.

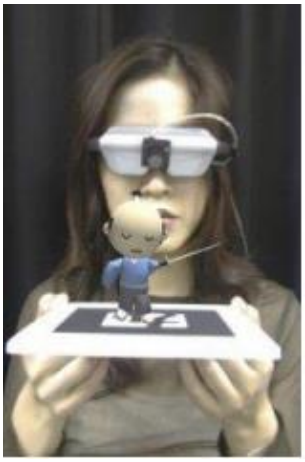

Figura 2: ARToolkit
Fuente:(Reinoso, 2014)

En 2004, investigadores de la Universidad Nacional de Singapur publicaron Human Pacman, un juego que usa GPS y un sistema inercial para registrar la ubicación de un jugador. Los participantes son básicamente jugadores humanos, que llevan computadoras y sistemas de visión por la ciudad y reconocen el mundo, como se muestra en la Figura 3.

Figura 3: H-Pacman
Fuente: (Gonzáles Morcillos, Carlos; Fernández vallejo et al., 2012)

En 2008, Mobilizy creó Wikitude (Figura 4), una aplicación que aumenta la información del mundo real con datos obtenidos de Wikipedia. Inicialmente, solo se podía usar en teléfonos Android, pero ahora se puede descargar en otras plataformas (por ejemplo, iPhone).

Figura 4: Wikitude
Fuente:(Reinoso, 2014)

A principios de 2010, Adidas lanzó un conjunto de cinco zapatillas deportivas de realidad aumentada (Figura 5). Los propietarios de estos modelos pueden transmitir contenido exclusivo enseñando la lengüeta del zapato a una cámara, donde se mostrará el mundo 3D de la marca, permitiendo al usuario visualizar contenido exclusivo y obtener premios especiales.

Figura 5: Zapatillas Adidas Originales, Augmented Reality
Fuente:(Gonzáles Morcillos, Carlos; Fernández vallejo et al., 2012)

Actualmente, la Realidad Aumentada vive su época dorada gracias al crecimiento explosivo de los smartphones, evolucionando hacia aplicaciones más fáciles de usar, más prácticas y útiles desde el punto de vista del usuario.

4.2. Realidad Aumentada Y La Educación

La educación interactiva es un proceso en el que los profesores enseñan a los estudiantes a participar activamente haciendo preguntas y sugerencias para mejorar. La educación interactiva es especialmente importante cuando se adquieren conocimientos de materias científicas prácticas a través de la metodología científica.

La AR se ha convertido en una tecnología utilizada en diversos entornos de la sociedad en los últimos años, y tiene un gran impacto en entornos relacionados con la publicidad, el marketing, los videojuegos y la educación principalmente.

Montecé Mosquera et al. (2017) afirman "que la realidad aumentada puede identificarse como la información adicional obtenida mediante la observación del entorno con el objetivo de brindar a los usuarios interacción, entretenimiento, motivación para aprender e innovar dentro del programa".

La AR se involucra activa y conscientemente en el desarrollo de habilidades cognitivas y espaciales debido a que permiten la validación, crítica, extensión y generación de ideas, sentimientos y opiniones a partir del conocimiento.

Con la realidad amentada podemos aprender de la observación fuera del aula, es un concepto diferente del aprendizaje basado en el descubrimiento. Del mismo modo es una herramienta muy útil dentro de la clase, ayuda a reforzar los conocimientos mediante la visualización de modelos 3D (Bohórquez Melo, 2018).

El uso de la realidad aumentada en el campo de la educación es generalizado y diverso, extendiéndose a diferentes niveles de aprendizaje para satisfacer las necesidades de profesores y estudiantes.

Entre los muchos valores agregados que la realidad aumentada puede aportar a la educación, podemos mencionar:(Sevilla, 2017)

- Motivación, el uso de una tecnología innovadora en el aula hace que la motivación del alumno aumente enormemente.
- Trabajo colaborativo, la realidad aumentada a través de sus aplicaciones y el uso de los dispositivos necesarios representa un recurso muy apropiado para realizar actividades entre alumnos, facilitando el trabajo en grupo.

- Construcción del conocimiento por parte del alumno, los estudiantes pueden aprender a explorar y participar en procesos paso a paso, guiados en la gestión de aplicaciones de realidad aumentada.
- Mayor información, a través de una imagen, un código, una palabra o un breve texto se accede a una gran información que amplía la posibilidad de adentrarse en conocimientos que con solo observarlos no sería posible.
- Tecnología gratuita, no es necesario realizar una gran inversión para poner la tecnología en práctica. La mayoría de las aplicaciones son gratuitas o tienen una versión gratis que pueden manejar funciones básicas y hacer grandes cosas.
- Mayor accesibilidad, es muy común que los estudiantes tengan en sus manos un dispositivo que puede ejecutar y usar la realidad aumentada, ya que hay muchos usuarios que disponen de teléfonos inteligentes y tabletas simples.
- Desarrollo de destrezas tecnológicas, usar la realidad aumentada en la educación es simple, pero permite un aprendizaje y manejo de la tecnología, que con recursos tradicionales no se podría lograr.

4.3. Realidad Aumentada

Según Rigueros (2017) la realidad aumentada consiste en combinar el mundo real con el virtual mediante un proceso informático, enriqueciendo la experiencia visual y mejorando la calidad de comunicación.

La realidad aumentada es una tecnología que integra señales recibidas en el mundo real (generalmente video y audio) con señales generadas por computadora (objetos gráficos 3D). Construye un mundo nuevo, mejorado y cohesionado que reacciona a los objetos del mundo real y virtual que coexisten en el ciberespacio.

En otras palabras, la realidad aumentada permite agregar información virtual a nuestro entorno físico del mudo real, creando una realidad mixta en tiempo real y favoreciendo a la perspectiva física del usuario, esto a través de dispositivos tecnológicos.

Los sistemas de realidad aumentada generalmente comienzan grabando las señales de audio y video, estas son procesadas por el sistema para mejorar los bordes del objeto y preparar la imagen para la división, extracción, reconocimiento de patrones y marcas. Utiliza este proceso para determinar dónde se debe reemplazar el objeto real

por el objeto virtual, la ubicación y la perspectiva que se debe colocar en el espacio real (espacio de video).

La realidad aumentada está relacionada con la tecnología de realidad virtual popular en la sociedad. Hay una serie de características comunes, incluida la incorporación de modelos virtuales gráficos 2D y 3D al campo de visión del usuario. La principal diferencia es que la realidad aumentada no transforma el mundo real en un mundo virtual, sino que mantiene el mundo real visto por el usuario superponiendo información virtual sobre el mundo real. Los usuarios pueden interactuar con información virtual superpuesta al mismo tiempo, sin perder el contacto con el mundo real a su alcance.

La RA ha demostrado ser una tecnología multidisciplinar que con los años ha irrumpido en diferentes ámbitos de aplicación, en tal sentido que la RA se encuentra omnipresente en nuestra vida cotidiana, siendo acogida con altos grados de aceptación entre los usuarios y compatible con diversos dispositivos tecnológicos. Hay gran coincidencia entre varios autores a la hora de afirmar que el sistema de RA es secuencial y ejecuta cuatro tareas que se describen a continuación(Ruiz Henry et al., 2018).

4.4. Tipos de realidad aumentada

4.4.1. Realidad aumentada geolocalizada.

Este tipo de aplicación es utilizada por los usuarios para buscar y acumular información sobre puntos de interés de su entorno. El uso eficaz de estos navegadores requiere el uso del hardware de geolocalización del dispositivo, incluidos GPS, brújula y acelerómetro, así como elementos de captura y proyección.

- GPS: Para detectar la ubicación en la que se encuentra el usuario.
- Brújula: Para revelar la orientación a la que mira el usuario.
- Acelerómetro: Para detectar la elevación y el ángulo

Según Castillo (2017) el modo en el que los usuarios hacen uso de este tipo de navegador AR es mediante la iniciación de la aplicación móvil y de la cámara

del dispositivo, enfocándolo en la dirección deseada para que de esta forma sea localizado el punto exacto en el que se encuentra el usuario y posteriormente se presente la información sobre el entorno que lo rodea.

4.4.2. Realidad Aumentada Basada en Marcadores.

La tecnología se basa en el uso de imágenes, logo, o cualquier objeto 2D que puede ser interpretado por un dispositivo móvil para brindar información adicional.

Usa marcadores, los cuales pueden estar superpuestos en símbolos o imágenes, el software de realidad aumentada puede seguir modelos o marcadores y ajustar la posición del modelo 3D que se muestra en la pantalla a medida que mueve o gira el modelo.

La finalidad de un marcador es incorporar la RA y para ello necesita de los siguientes elementos(Castillo Otegui, 2017):

- Elementos de captura: La cámara de un Smartphone, de una Tablet, de unas gafas de RA, etc.
- Elemento de proyección: Necesita de una pantalla donde poder proyectar la información que suministra el marcador.
- Elemento de proceso: Hace falta una aplicación o programa de software que ejecute la información suministrada por el marcador.
- Activadores: Se trata de las imágenes, objetos, códigos QR, etc. que activan la aplicación para que ésta descargue el resultado final.

El primer paso es encontrar marcadores en revistas, libros, periódicos, productos y más. Si lo encuentra, el segundo paso es abrir una aplicación que la cámara ejecuta automáticamente para buscar el disparador. Luego la cámara enfoca el activador de tal manera que lo reconoce y descarga la información contenida como resultado final. Puede incluir cualquier cosa, desde imágenes simples y texto informativo hasta elementos más complejos como objetos 3D, videos y videojuegos.

4.4.3. Realidad aumentada basada en reconocimiento de imágenes.

Implementa contenido virtual utilizando una imagen de entorno como disparador, en lugar de patrones y marcadores, el disparador es la imagen en sí, por lo que no hay elementos intrusivos en la escena.

4.5. Niveles de realidad aumentada.

Los niveles de realidad aumentada se definen por los diferentes niveles de complejidad que presentan las aplicaciones basadas en realidad aumentada, en función de la tecnología implementada. (Sevilla, 2017).

4.5.1. Nivel 0 (enlazados con el mundo físico).

Se basa principalmente en el uso de códigos de barras UPC (Código de producto universal) y QR (Respuesta rápida), los cuales son hipervínculos a otro contenido.

Debido a que los códigos QR son diferentes de los marcadores de realidad aumentada, solo pueden ser identificados por la aplicación en la que fueron creados. La información que se muestra en los marcadores de realidad aumentada depende de la aplicación involucrada. Sin embargo, con un código QR, la información está codificada en el propio símbolo, por lo que cualquier lector de códigos QR puede leerla.

Los activadores en este nivel son los códigos QR que enlazan con sitios web. Un código QR es un módulo para almacenar información en una matriz de puntos o en un código de barras bidimensional (Figura 6) y fue creado en 1994 por la compañía japonesa Denso Wavew(Bohórquez Melo, 2018).

Figura 6: Código QR
Fuente: (Bohórquez Melo, 2018)

4.5.2. Nivel 1 (AR con marcadores).

Las aplicaciones de este nivel suelen utilizar marcadores o imágenes y diagramas trazados para el reconocimiento de patrones en 2D, pero también tiene la posibilidad de reconocer objetos 3D como cubos con marcadores en varias caras.

Los marcadores de referencia son unos objetos utilizados para la observación de sistemas de imágenes, los cuales aparecen en la imagen para ser usados como punto de referencia o de medida. En este nivel los activadores son marcadores (Figura 7), figuras que cuando son escaneadas normalmente se convierte en un modelo 3D que se superpone en la imagen real. Los marcadores necesitan un patrón único, el cual le permitirá a la cámara reconocer y determinar el objeto u objetos que debe mostrar(Bohórquez Melo, 2018).

Figura 7: Marcador de realidad aumentada
Fuente:(Bohórquez Melo, 2018)

4.5.3. Nivel 2 (AR sin marcadores).

La aplicación reemplaza el uso de marcadores por el GPS del dispositivo móvil y la brújula para determinar la posición y orientación del usuario y superpone puntos de interés sobre la imagen real.

En este nivel, como su nombre lo indica, se basa en reconocer elementos que actúan como disparadores, como objetos e imágenes o localizaciones GPS. Este nivel también tiene la geolocalización del usuario por ubicación y orientación.

Desde el 2009 se han venido desarrollando aplicaciones para dispositivos móviles llamadas navegadores de realidad aumentada; estas aplicaciones utilizan el hardware de los teléfonos inteligentes (GPS, brújula y acelerómetro) para localizar y superponer una capa de información sobre puntos

de interés de nuestro entorno (Figura 8). Cuando el usuario mueve el teléfono inteligente captando la imagen de su entorno, el navegador, a partir de un mapa de datos, muestra los puntos de interés (POIs) cercanos(Bohórquez Melo, 2018).

Figura 8: Realidad aumentada con geolocalización
Fuente:(Bohórquez Melo, 2018)

En este nivel de aplicación, cuando se reconoce una imagen u objeto, se activa una pila de contenido digital (imagen, video o modelo 3D). La imagen tomada con la cámara del teléfono inteligente se compara con la imagen de referencia y, si coinciden, el objeto virtual se muestra en el mundo real.

Desde la llegada de los primeros navegadores de realidad aumentada, se han desarrollado muchas aplicaciones basadas en geolocalización. La aplicación permite a los usuarios ver lo que sucede a su alrededor y acceder a información sobre su entorno, como sitios históricos entre otras cosas.

4.5.4. Nivel 3 (Visión aumentada).

Este nivel contempla la utilización de gafas específicas Google Project glass o de lentillas biónicas. Este último nivel aún no está en el mercado a nivel comercial, es por eso que se dice que será el futuro de la realidad aumentada.

La realidad aumentada está integrada en las gafas (Figura 9), lo que permite a los usuarios ver la información disponible y acceder a Internet a través de comandos de voz sin usar las manos.

Figura 9: Realidad aumentada con Google Glass
Fuente:(Bohórquez Melo, 2018)

Así mismo en la Universidad de Washington se están desarrollando lentes de contacto (figura 10) que proyectan la realidad aumentada directamente en nuestros ojos, por lo que es considerado uno de los prototipos más innovadores en el campo de la realidad aumentada hasta el momento.

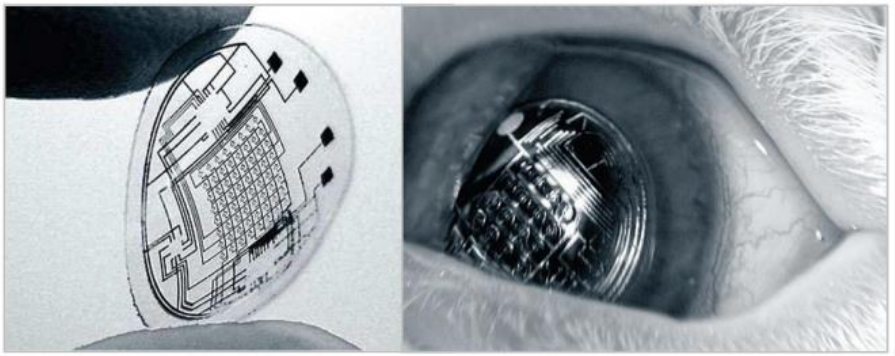

Figura 10: Lentes biónicas de AR
Fuente:(Álvarez, 2017)

4.6. Proceso de la realidad aumentada

El proceso de creación de realidad aumentada es fácil de entender e incluye elementos como cámaras, software y dispositivos con obturadores.

Al disponer de un dispositivo con un software instalado previamente el primer paso sería activar la aplicación en cuestión, enfocar con la cámara del dispositivo la realidad física sobre la que queremos obtener la información adicional y capturarla. De forma inmediata y tras la transformación de los datos por parte de la aplicación o software, la pantalla del dispositivo mostrará la información adicional que conlleva asociada la realidad que ha sido capturada por la cámara(Sevilla, 2017).

El sistema de realidad aumentada requiere al menos cuatro tareas o pasos básicos para completar el proceso de aumento, como la captación de escena, identificación de escena, mezclado de realidad y visualización.

En la siguiente figura se muestra las fases citadas para llevar a cabo el proceso de realidad aumentada.

Figura 11: Fases del proceso de RA
Fuente:(Álvarez, 2017)

4.6.1. Captación de la escena.

El primer paso en el proceso de realidad aumentada es capturar a través de un dispositivo de visualización capaz de reconocer y procesar la escena para comprender el entorno físico y procesarlo. El dispositivo de captura de imágenes es un dispositivo físico de recopilación de realidad que desea ser ampliada.

Estos dispositivos se pueden agrupar principalmente en dos grupos:

- **Dispositivos video-through:** dispositivos que realizan la captura de imágenes o video y están aislados de los de visualización. En este conjunto se encontrarían las cámaras de video o los terminales móviles que dispongan de una cámara integrada(Álvarez, 2017).
- **Dispositivos see-through:** son los dispositivos que realizan tanto la tarea de capturar la escena real como de mostrarla con información aumentada al usuario. Estos dispositivos acostumbran a trabajar en tiempo real. Dentro de este grupo se encontrarían aquellos dispositivos

conocidos como Head Mounted Display (HMD) y Head Up Display (HUD), ambos desarrollados y utilizados en la actualidad en el ámbito militar(Álvarez, 2017).

4.6.2. Identificación de la escena o el entorno.

"La siguiente fase o tarea que se realiza dentro del proceso de realidad aumentada consiste en analizar y averiguar qué entorno o escenario real, a elección del usuario, se va a aumentar junto con la información digital"(Álvarez, 2017).

Para este proceso de reconocimiento es necesario especificar el escenario físico que va a ser extendido con el sistema de información virtual y realiza un proceso de reconocimiento de objetos virtuales para identificar el objetivo o el marcador de la imagen.

4.6.3. Aumento de la realidad.

Tras el proceso de identificación de la escena o del entorno, el siguiente paso que tiene lugar dentro de los sistemas de realidad aumentada es el de incluir la información digital que se quiere aumentar, sobre la escena real previamente capturada(Álvarez, 2017).

La información digital que se puede mostrar tanto de forma visual (imagen y texto), táctil como auditiva, a pesar de que la mayoría de los sistemas de realidad aumentada solo proporcionan información de carácter visual. Este proceso es el tipo de información que necesitamos ampliar. Puede distinguir dos tipos de información: bidimensional (2D) y tridimensional (3D).

4.6.4. Visualización.

Finalmente, el proceso de última etapa que tiene lugar en un sistema de realidad aumentada es una visualización que incluye renderizado del entorno real, así como información virtual agregada a través del dispositivo de visualización.

En esta sección se realiza una clasificación entre sistemas comunes y específicos según su coste, utiliza sistemas genéricos o de bajo costo y sistemas específicos o de alto costo.

4.7. Agentes relevantes en la provisión de servicios de realidad aumentada

Para poder brindar los servicios de realidad aumentada son necesarios varios agentes tales como: los fabricantes de los dispositivos, los desarrolladores del software de realidad aumentada, los datos y finalmente la red proporcionada por los operadores de telecomunicaciones.

4.7.1. Fabricantes de Dispositivos.

Uno de los actores más importantes para los usuarios es el fabricante de hardware, es decir, el fabricante de dispositivos que puede acceder a este tipo de servicios. Los fabricantes de estos smartphones, PCs, tablets, consolas y, en el caso de la realidad aumentada, gafas especiales o cascos que integran monitor y proyector, permitirán ver en directo la futura comunicación digital integrada.

4.7.2. Desarrolladores del software de realidad aumentada.

Se trata de los agentes que desarrollan las aplicaciones que dotan de vida a los datos que complementan la visión del entorno físico y gracias a las cuales es posible ver de una forma aumentada el mundo y así enriquecer los objetos de la vida real (personas, lugares o cosas) o realizar un seguimiento e interactuar con ellos de una manera radicalmente diferente(Fundación Telefónica, 2011).

Estos agentes, por un lado, son los encargados de proporcionar las herramientas adecuadas para habilitar la realidad aumentada. Por ejemplo, en este sentido, existen aplicaciones que soportan el reconocimiento visual, procesamiento de imágenes, u otras aplicaciones basadas en marcadores, por otro lado, la información digital almacenada del mundo para el uso posterior a través de la lente de realidad aumentada. Al coordinar la información, brinda a las personas la posibilidad de interactuar con el mundo en diferentes dimensiones al mismo tiempo que contribuye con datos propios para la construcción de este mundo digital paralelo y que complementa a la información del mundo real.

4.7.3. Mundo digital y digitalización.

Los actores más relevantes son los datos, el mundo digital y la digitalización. En los últimos años se ha recopilado mucha información sobre el mundo real: datos de ubicación, descripciones, imágenes, etc. La información sobre lugares, cosas, personas y todo lo demás se almacena en la nube, formando un mundo

digital paralelo. Un ejemplo particular es Google Earth, la base de datos más grande del mundo de imágenes de satélite, mapas, relieves y edificios.

Los proveedores de contenido como empresas (que se especializan en diversos campos como entretenimiento, aprendizaje en línea, mundos virtuales, etc.) o por los propios usuarios pueden generar datos digitales.

4.7.4. Red: operadores de telecomunicaciones.

El otro elemento de esencial importancia para que estas aplicaciones puedan llegar al público es la red, precisamente porque la potencia de la realidad aumentada radica en poder acceder a la información digital complementaria a la del mundo físico actualizada en tiempo real(Fundación Telefónica, 2011).

Esta información suele ser accesible en cualquier momento y en cualquier lugar, como en aplicaciones de referencia geográfica. Por estas razones, los operadores deben proporcionar una amplia gama de conexiones de datos para utilizar este tipo de aplicaciones, lo que juega un papel particularmente importante en este ecosistema. Como se mencionó anteriormente, la nube debe proporcionar los servicios de conectividad necesarios para almacenar datos y proporcionar acceso al servicio RA.

4.8. Mapa topográfico

Antes de adentrarnos al concepto de mapa topográfico es importante conocer el concepto de cartografía. Si bien menciona Pino & Calderón (2017) que la cartografía es la ciencia que trata de recopilar y analizar medidas de las distintas regiones del planeta para su posterior representación gráfica en distintas dimensiones o escalas.

En otras palabras, la cartografía es la ciencia de medir, recopilar y analizar datos en muchas regiones de la Tierra y trazarlos en otras dimensiones lineales (escala reducida) o también se conoce comúnmente como un conjunto de documentos territoriales que se refieren a un campo de estudio en particular.

Los mapas topográficos representan la superficie de la tierra utilizando curvas de nivel que muestran cambios en el relieve de la tierra. Además de las curvas de nivel, a menudo contiene otras variables geográficas como vegetación, suelo, red hidrográfica, localidades, etc., todas las cuales tienen sus propios colores y símbolos.

Según el Instituto Geografico Nacional del Gobierno de España (2017) un mapa topográfico o de propósito general es el que representa gráficamente los principales elementos que conforman la superficie terrestre, como vías de comunicación, entidades de población, hidrografía, relieve, con una precisión adecuada a la escala.

Los mapas topográficos pueden mostrar una variedad de características naturales y artificiales creadas por el ser humano, desde el curso de agua, hasta carreteras o edificios. Los elementos que se muestran pueden variar de un mapa a otro, dependiendo de las características en las que se desarrolló cada uno de los mapas topográficos.

En un mapa topográfico, no se representan todas las curvas de nivel, se representan únicamente las que corresponden a unas altitudes determinadas. Esas altitudes son arbitrarias y vienen definidas por el tipo y escala del mapa que vayamos a utilizar. A la diferencia de altitud de una curva de nivel con respecto a otra le denominamos equidistancia, así como se muestra en la figura 12(Robles & Escudero, 2017).

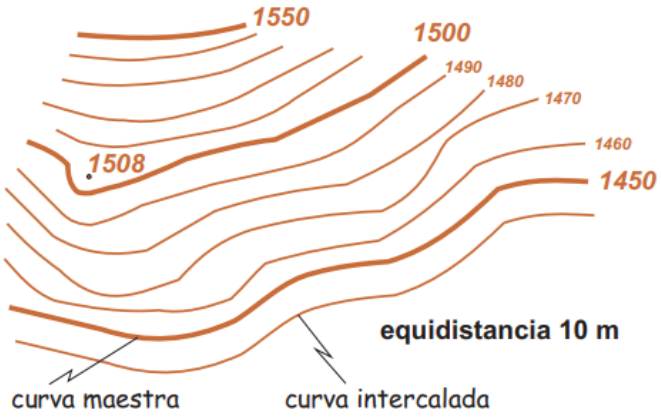

Figura 12: Representación de equidistancia.
Fuente:(Robles & Escudero, 2017)

Existen características que ayudan a definir un mapa topográfico y que permiten una representación adecuada de la superficie.

4.8.1. Curvas de nivel.

Este es el método más utilizado para representar gráficamente la forma del relieve de la superficie del terreno. Esto se debe a que es una forma fácil y rápida de dibujar un perfil, determinando la cota o elevación de cualquier punto del

terreno, así mismo calcular pendientes, resaltar las formas y características del terreno.

Según Leonardo Casanova (2010) una curva de nivel es el trazado que la superficie del terreno marca sobre un plano horizontal que la intercepta, por lo que podríamos definirla como la línea continua que une puntos de igual cota o elevación. Este concepto se ilustra en la figura 13.

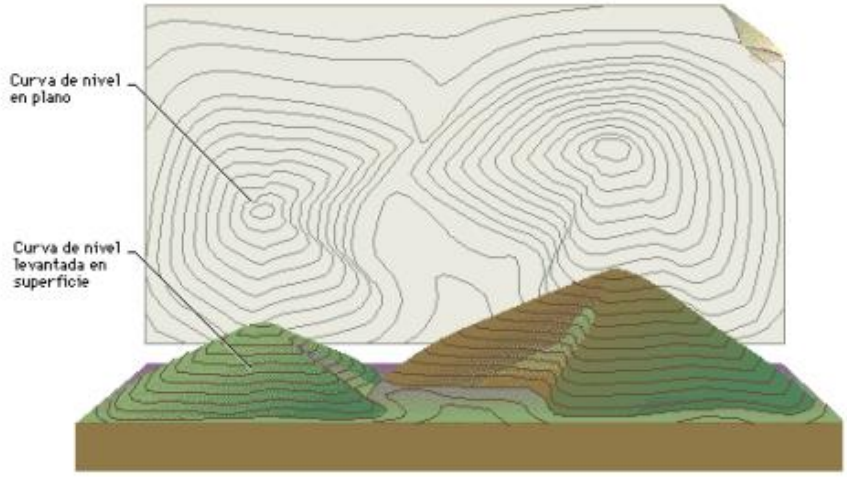

Figura 13: Representación del concepto de curvas de nivel
Fuente:(Casanova, 2010)

Hay dos tipos de curvas de nivel: curvas maestras y curvas intercaladas. La curva maestra es una curva que se muestra como una línea más gruesa por cada cinco líneas de contorno. Estas curvas le permiten resaltar las curvas de nivel restantes y filtrar la información, especialmente en áreas con pendientes muy pronunciadas, y las curvas de nivel aparecen tan cerca que puede visualizar la información rápidamente. Las curvas intercaladas son los trazos más finos de los contornos que aparecen entre los contornos principales.

4.9. Modelos Geográficos

Consiste en una representación en dos o tres dimensiones de un volumen de terreno. Esto se puede representar la mineralización, litología, alteración u otro tipo de característica geológica(Vivas et al., 2020).

El modelado simplifica problemas más complejos relacionados con la realidad, esto permitirá comprender mejor cómo funciona el sistema. Hay tres tipos de modelos utilizados en geología. Modelo geológico que muestra la distribución de materiales, sus características geográficas, sus características hidrológicas y geomorfológicas. El

modelado geomecánico distingue entre propiedades geotécnicas e hidrogeológicas de los materiales, y el modelo numérico es una herramienta diseñada para mostrar versiones simplificadas de la realidad.

4.10. Modelos Hidrológicos

Un modelo hidrológico es una representación simplificada de un sistema real en forma física o matemática. Matemáticamente, el sistema se expresa como una expresión analítica. De esta forma, los modelos matemáticos ayudan a tomar decisiones relacionadas con la hidrología. Por lo tanto, se requiere conocimiento de las entradas y salidas del sistema.

La aplicación de los modelos radica, entre otros aspectos, en la simulación y predicción de los fenómenos físicos a corto, mediano y largo plazo. Asimismo, a través de la modelación podemos encontrar relaciones causa efecto. Algunos modelos observan las escalas de tiempo, mientras que otros pueden usar medidas a corto tiempo y simular un lapso de tiempo relativamente corto como en el caso de una inundación repentina(Vivas et al., 2020).

5. Materiales y Métodos

Para el desarrollo de la siguiente investigación se aplicó varios métodos y herramientas que permitieron el cumplimiento satisfactorio de todas las actividades que se planificaron.

5.1. Materiales

Para la elaboración y ejecución del diseño e implementación del simulador de mapa topográfico interactivo se utilizaron varios materiales tanto para el diseño de la caja de arena como para la implementación del software.

5.1.1. Materiales de software

5.1.1.1. Paquete ARSandbox

Dentro de ARSandbox se encuentran paquetes informáticos como Vrui VR, Kinect 3D video capture Project y sandbox, que se distribuyen de forma libre bajo la licencia GNU General Public License en el portal del desarrollador.

Vrui VR es kit de herramientas que nos permite admitir aplicaciones totalmente estables y portátiles que se ejecutan en una variedad de entornos de realidad virtual, comenzando desde una computadora portátil con un panel táctil y avanzando hacia entornos de escritorio con dispositivos de entrada especiales.

La tarea del kit de herramientas se enfoca en dos áreas principales de ayuda para los desarrolladores, las interacciones y las técnicas de locomoción.

5.1.1.2. Linux

Para la implementación del presente proyecto se hizo uso del sistema operativo Linux, específicamente del sistema Linux Mint versión 19.3, debido a que este es un sistema estable, seguro y lo más importante es un sistema operativo de código abierto, el cual permite mayor manipulación y una fácil instalación del software sandbox de realidad aumentada.

Dentro de las grandes ventajas del sistema operativo Linux, se debe a que es un sistema operativo multiusuario, multitarea de libre distribución y código abierto permitiendo la incorporación de diferentes gestores de ventanas, además de la posibilidad de poder configurarlo dependiendo de las necesidades del usuario.

Una de las características que personalmente es importante de este sistema operativo es la fácil instalación de todos los recursos que se necesitan para

el ARsandbox, debido a que mediante la terminal se pueden descargar e instalar cada uno de ellos, con insertar varias líneas de código en la misma.

5.1.1.3. Solidworks

El siguiente software se utilizó para el desarrollo del diseño del dispositivo sandbox de realidad aumentada. Siendo este uno de los softwares más completos y al cual estamos más familiarizados para tener un mayor control y versatilidad al momento de la creación y edición del modelo de la caja de arena como tal, esto con la finalidad de adaptarlo a las necesidades del usuario y tratando de adaptarnos a los materiales disponibles en el medio.

SOLIDWORK es un software de diseño en 3D para moldear y ensamblar piezas, además puede proveer de diferentes herramientas que permitirá gestionar y controlar de una forma segura los datos.

También se utilizó para diseñar los perfiles de cada una de las partes de la estructura de la caja de arena, para luego poder ensamblarlas y que encajen de una manera correcta.

5.1.2. Materiales de Hardware

5.1.2.1. Computadora

Tomando en cuentas las consideraciones expresadas por Oliver Kreylos se deberá contar con una computadora con una tarjeta gráfica de alta gama, que ejecute cualquier versión de Linux, con un chip gráfico, que podría ser una tarjeta Nvidia GeForce y un procesador por encima de Core i3 7ma generación. La PC debe tener una buena CPU, pero hay cierta libertad en cuanto a la RAM, debido a que 2GB son suficientes para ejecutar el software AR Sandbox o en cuanto almacenamiento, un disco duro de 20GB son suficientes para instalar Linux y el software AR Sandbox.

Figura 14: computadora portátil
Fuente: (Magitech, 2020)

Es importante mencionar que para poder generar el mapa topográfico de la mejor manera se requiere que el procesador y la tarjeta de video cumplan con las características especificadas. Sin embargo, en nuestro caso se utilizó una laptop DELL Inspirion i7 7500u, la cual cuenta con las siguientes características que son adecuadas para la implementación.

- Intel Core i7 7500U 2.7GHz - Séptima Generación.
- RAM: 24 GB DDR4 / Disco Duro 1TB
- Pantalla LED 15,6" HD 1366x768
- Video AMD Radeon R5 M445 4GB ddr5 dedicados
- DVD-RW, Teclado iluminado, Bluetooth 4.0.
- Windows 10 Home

5.1.2.2. Cámara Microsoft Kinect 3D y Adaptador USB a 2.0

Este dispositivo es una cámara de profundidad, con una percepción en 3D que permite capturar el movimiento de las personas de manera eficiente, mientras la cámara infrarroja que incluye permite diferenciar la profundidad. El adaptador USB a 2.0 permite la conexión del dispositivo Kinect 3D a la computadora mediante el puerto USB.

Para el presente proyecto se utilizó un sensor Kinect de la primera generación 1414 que permite capturar el movimiento de las personas a través de más de 48 puntos de articulación, mediante un algoritmo bastante complejo de reconocimiento de imágenes.

Es utilizado para la generación del mapa topográfico, recopila la información del relieve moldeado en la caja de arena que será procesado por el software AR sandbox y posteriormente proyectada en dicha superficie.

Figura 15:Kinect y adaptador
Fuente: Autor

5.1.2.3. Proyector de video digital

Este dispositivo nos permitirá proyectar e interactuar con el mapa topográfico generado en la superficie de la arena por el software ARSandbox. En nuestro caso se ha utilizado un proyector de alcance corto y una relación de aspecto nativa de 4:3 para que coincida con el campo de visión de la cámara Kinect.

En este caso se escogieron estas características debido a que este tipo de proyectores de alcance corto nos permite acoplar las distancias de proyección y escaneo de la superficie de la arena.

Figura 16: proyector de video digital
Fuente: (Epson, 2021)

5.1.2.4. Caja de arena

Se utilizo una caja de arena que se adapta a la distancia de detección mínima y máxima de la cámara Kinect y por ende a la resolución deseada. El tamaño adecuado de la caja de arena será de 40”x30” o 1m x 75 cm.

Para una mayor movilidad se optó por realizar un diseño que posea ruedas, así mismo la altura de la caja de arena será de aproximadamente 20 cm.

En cuanto al material para su elaboración, se creyó conveniente la utilización de madera, por la facilidad de adquisición en el medio y la cómoda manipulación que esta presenta al momento de la construcción de la caja.

Figura 17: Caja de arena
Fuente:(Worthington Direct, 2021)

5.1.2.5. Estructura de aluminio

Se elaboró una estructura de aluminio posicionada en la parte céntrica de la caja para poder ubicar la cámara Kinect y el proyector, estos tendrán que ir ubicados a una distancia determinada para que el escaneo y la proyección se acople a las dimensiones de la caja de arena, siendo exactos a 1 metro de distancia de la superficie de arena.

Figura 18: Estructura de aluminio
Fuente:(Worthington Direct, 2021)

5.1.2.6. Arena

La caja de arena debe llenarse con arena a una profundidad de alrededor de 4” o 10 cm, La arena que se debería emplear es la arena sandtastik que posee

excelentes propiedades de proyección para que la imagen proyectada sobre ella se pueda visualizar de una manera adecuada y la cantidad de la misma será de 200 lb aproximadamente.

Esta arena debe ser de color blanco o amarillo para mayor claridad de la imagen proyectada y debe ser hipoalergénica para que no produzca alguna sustancia al momento de su manipulación.

Sin embargo, por cuestiones de accesibilidad de este tipo de arena, se optó por implementar un tipo de arena con características similares, debido a que el tipo de arena mencionado en un principio es de difícil adquisición en el país.

5.1.3. Presupuesto

El presupuesto total para el desarrollo e implementación del proyecto se resume en la tabla 1, en la cual se describen todos los componentes físicos que se deben adquirir para la elaboración del simulador de mapa topográfico interactivo.

Tabla 1*: Costos de materiales e insumos*
Fuente: Autor

Elemento	Unidades	Precio por unidad	Costo
Computadora	1	$600	$600
Cámara Microsoft Kinect 3D	1	$60	$60
Proyector de tiro corto	1	$650	$650
Caja de Arena	1	$160	$160
Arena (200 lb)	1	$50	$50
Gastos varios e imprevistos	1	$25	$25
TOTAL			$1545

Cabe mencionar que algunos de estos elementos como computadora y proyector van a ser proporcionados por la carrera de Minas de la Facultad de la Energía de la UNL, sin embargo, se ha tomado en cuenta dentro del presupuesto para tener

una noción del gasto total que conllevaría la implementación del proyecto, el financiamiento de resto de materiales será cubierto por el responsable del proyecto.

5.2. Métodos

5.2.1. Método Incremental

Este método incremental se implementó para el desarrollo de cada una de las etapas de construcción del simulador de mapa topográfico interactivo, desde las fases de diseño hasta poder culminar su construcción.

5.2.2. Método inductivo

Este método permitió realizar una indagación de la información de proyectos o investigaciones que se desarrollaron con anterioridad para poder guiar la elaboración del simulador de mapa topográfico, en cuanto a diseño y software se refiere.

5.2.3. Método deductivo

El método deductivo se utilizó para seleccionar los equipos y software necesarios como el tipo de arena, medidas de la caja de arena, ubicación de los equipos de obtención de información y de proyección, todo esto basado en los criterios de diseño y características de los equipos utilizados.

5.2.4. Método analítico

Mediante el uso del método analítico se realizó la valoración de los beneficios que presenta las características de la arena para poder realizar un modelado adecuado de la superficie y poder simular la mayor cantidad de relieves.

5.3. Especificaciones de la metodología

La metodología especificada con anterioridad se realizó en 3 fases, las cuales permiten cumplir con los objetivos planteados, a continuación, se detallarán cada una de estas fases.

5.3.1. Fase conceptual

En esta fase se recopila toda la información y detalles del software que permitirá la proyección del modelo de realidad aumentada sobre la superficie. Al mismo tiempo se analiza las características de cada uno de los componentes, respetando

las recomendaciones que se describen en cuanto a especificaciones técnicas para poder obtener un prototipo de excelente calidad.

5.3.2. Construcción de la estructura

Para la fase de construcción se toma en cuenta las medidas recomendadas por el creador original del prototipo, tomando en cuenta que estas ya fueron probadas para una correcta proyección en la superficie de la arena. También cabe recalcar que se hicieron ciertas modificaciones en cuanto a la altura de la caja de arena y dispositivos adicionales para su movilidad, sin embargo, esto no afecta a su funcionamiento.

5.3.3. Calibración del prototipo

Para la fase de calibración se realizaron varias pruebas para corregir el ángulo de proyección del proyector y la recopilación de los puntos de la superficie de arena que son captados mediante el sensor, esto con la finalidad de brindar la mejor experiencia al usuario.

5.4. Descripción General

En la figura 19 se muestra el proceso que lleva a cabo el software, el cual consta de 4 etapas principales, las mismas que serán descritas a continuación.

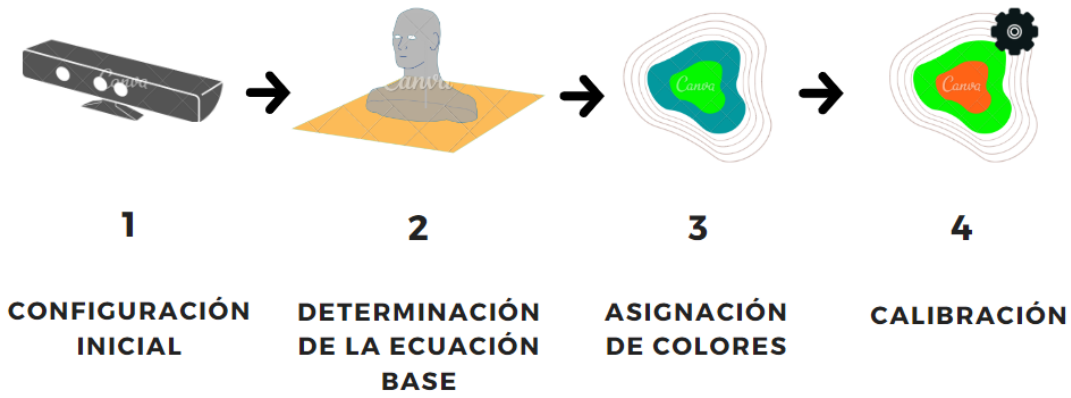

Figura 19: Diagrama de funcionamiento
Fuente: Autor

5.4.1. Configuración inicial

En esta etapa se desarrolla la ejecución de la cámara Kinect, configurándola con los parámetros de calibración de fábrica y determinando la ecuación formula de conversión de profundidad en base a la distancia focal, altura real del objeto y la

altura de la imagen. Es importante mencionar que el código está desarrollado en el lenguaje de programación C++.

```
/*****************************
CONFIGURACIÓN INICIAL:
*****************************/

/* Abrir cámara kinect*/
        Kinect::Camera camera(kinectDevice.getDevice()); // Elige automáticamente entre los dispositivos "predeterminados" para un sistema operativo
std::cout<<"Downloading factory calibration data for Kinect "<<camera.getSerialNumber()<<"..."<<std::endl; // envía datos a la consola para ser impresos como texto
/*  Recuperar los datos de calibración de fábrica: */
        Kinect::Camera::CalibrationParameters cal;
        camera.getCalibrationParameters(cal);// Devuelve los parámetros de calibración.

/* Calcula la fórmula de conversión de profundidad */
 //en base a la distancia focal, altura real del objeto(mm) y altura de la imagen(píxeles)
        double numerator=10.0*(4.0*cal.dcmosEmitterDist*cal.referenceDistance)/cal.referencePixelSize;
        double denominator=4.0*cal.dcmosEmitterDist/cal.referencePixelSize+4.0*cal.constantShift+1.5;
        std::cout<<"Depth conversion formula: dist[mm] = "<<numerator<<" / ("<<denominator<<" - depth)"<<std::endl;
```

El sensor Kinect entrega imágenes de profundidad con una resolución de 640x480 pixeles y sus datos pueden representados en un espacio de color RGB (Red Green Blue) en 16 bits.

Mediante la función depth se logra obtener el valor de profundidad, la cual trabaja bajo el método de luz estructura (figura 20), adquiriendo la información de profundidad por medio de la proyección de patrones regulares de puntos infrarrojos que se proyectan sobre un objeto, los cuales son captados por el sensor de profundidad infrarrojo. Posteriormente utiliza el proceso de triangulación para determinar la distancia de un punto en un espacio 3D a partir de la visión de dos o más imágenes, es decir los valores de disparidad en cada punto.

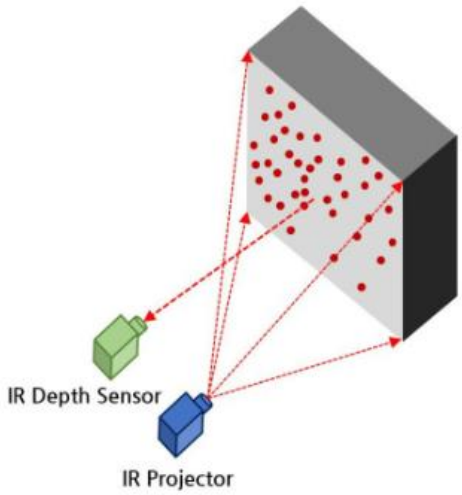

Figura 20: Método de luz estructurada
Fuente: (Kim & Choi, 2019)

```
/* Calcula la matriz de no proyección de píxeles de profundidad: */
   double scale=2.0*cal.referencePixelSize/cal.referenceDistance;
// la función Depth devuelve la profundidad de un elemento de la matriz
            Math::Matrix depthMatrix(4,4,0.0);
            depthMatrix(0,0)=scale;
            depthMatrix(0,3)=-scale*640.0/2.0;
            depthMatrix(1,1)=scale;
            depthMatrix(1,3)=-scale*480.0/2.0;
            depthMatrix(2,3)=-1.0;
            depthMatrix(3,2)=-1.0/numerator;
            depthMatrix(3,3)=denominator/numerator;

            {
std::cout<<std::endl<<"Depth unprojection matrix:"<<std::endl;
            std::streamsize oldPrec=std::cout.precision(8);
            std::cout.setf(std::ios::fixed);
            for(int i=0;i<4;++i)
                  {
                  std::cout<<"    ";
                  for(int j=0;j<4;++j)
                        std::cout<<' '<<std::setw(11)<<depthMatrix(i,j);
                  std::cout<<std::endl;
                  }
            std::cout.unsetf(std::ios::fixed);
            std::cout.precision(oldPrec);
            }

/* Calcular el mapeo de profundidad a color: */
            double colorShiftA=cal.dcmosRcmosDist/(cal.referencePixelSize*
   2.0)+0.375;
            double colorShiftB=cal.dcmosRcmosDist*cal.referenceDistance*1
   0.0/(cal.referencePixelSize*2.0);
```

```
31.             std::cout<<"Depth-to-color pixel shift formula: Shift =
    "<<colorShiftA<<" - "<<colorShiftB<<"/dist[mm]"<<std::endl;
32.
33. /* Fórmula para pasar directamente de profundidad al desplazamiento: */
34.             double dispA=colorShiftA-colorShiftB*denominator/numerator;
35.             double dispB=colorShiftB/numerator;
36.             std::cout<<"Or: Shift = "<<dispA<<" +
    "<<dispB<<"*depth"<<std::endl;
```

5.4.2. Determinación de la ecuación base

Para calcular el plano de referencia o plano base es necesario conocer las coordenadas y distancia de al menos tres puntos, los mismo que se proporcionan mediante la especificación de la superficie y de sus esquinas. La ecuación que representa el plano base es la siguiente:

$$Ax + By + Cz + D = 0$$

```
1.  /*************************************************************
2.  DETERMINACIÓN PARÁMETROS INTRÍNSECOS Y LECTURA DE
    ECUACIÓN BASE
3.  *************************************************************/
4.
5.  /* Obtener los parámetros intrínsecos de la cámara:  como distancia focal y
    punto principal*/
6.      cameraIps=camera->getIntrinsicParameters();
7.      /* Leer el archivo de diseño de sandbox: */
8.      Geometry::Plane<double,3> basePlane;
9.      Geometry::Point<double,3> basePlaneCorners[4];
10.     {
11.     IO::ValueSource layoutSource(IO::openFile(sandboxLayoutFileName.c
    _str()));
12.     layoutSource.skipWs();
13.
14.     /* Leer la ecuación del plano base: */
15.     std::string s=layoutSource.readLine();
16.     basePlane=Misc::ValueCoder<Geometry::Plane<double,3> >::decode(s.
    c_str(),s.c_str()+s.length());
17.     basePlane.normalize();
18.
19.     /* Lee las esquinas del cuadrilátero base y proyéctalas en el plano base:
    */
20.     for(int i=0;i<4;++i)
21.         {
22.         layoutSource.skipWs();
23.         s=layoutSource.readLine();
24.         basePlaneCorners[i]=basePlane.project(Misc::ValueCoder<Geome
    try::Point<double,3> >::decode(s.c_str(),s.c_str()+s.length()));
25.         }
```

```
26.     }
```

5.4.3. Asignación de colores

El software utiliza una combinación de varios shaders GLSL (lenguaje de sombreado OpenGL) para colorear la superficie por elevación usando mapas de color personalizables.

Este tipo de sombreado o shader conocido como sombreado de fragmentos (o texturas) definen los colores RGBA (Red Green Blue Alfa) para cada pixel que se procesa. Los colores RGBA está compuesto por rojo, verde, azul, alfa. Donde el parámetro alfa (figura 21) es un número entre 0.0 (totalmente transparente) y 1.0 (nada transparente) el cual es agregado en un proceso posterior dependiendo de la profundidad del pixel.

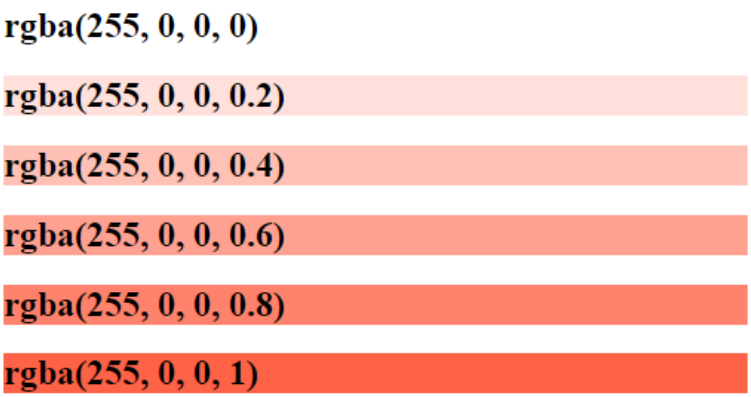

Figura 21: Efecto del parámetro alfa
Fuente: (W3Schools, 2022)

```
1.  /*******************************************************
2.  ASIGNACIÓN DE COLORES DEPENDIENDO DE LA PROFUNDIDAD
3.  *******************************************************/
4.
5.  void RawKinectViewer::mapDepth(unsigned int x,unsigned int y,float depth,
    GLubyte* result) const
6.          {
7.          if(depthPlaneValid)
8.                  {
9.  /* Colorear los píxeles de profundidad según la distancia al plano de
    profundidad: */
10.                 float dist=camDepthPlane.calcDistance(Plane::Point(float(x)+0.5f,f
    loat(y)+0.5f,depth));
11.                 if(dist>=0.0f)
12.                         {
13.                         GLubyte col=dist<depthPlaneDistMax?255U-
    GLubyte((dist*255.0f)/depthPlaneDistMax+0.5f):0U;
```

```
14.                     result[0]=col;
15.                     result[1]=col;
16.                     result[2]=255U;
17.                     }
18.             else
19.                     {
20.                     GLubyte col=-dist<depthPlaneDistMax?255U-GLubyte((-
    dist*255.0f)/depthPlaneDistMax+0.5f):0U;
21.                     result[0]=255U;
22.                     result[1]=col;
23.                     result[2]=col;
24.                     }
25.             }
26.         else
27.             {
28. /* Píxeles de profundidad de color según el valor de profundidad: */
29.             //Mediante un arreglo de 6x3
30.             static const GLubyte mapColors[6][3]=
31.                     {
32.                     {255,0,0},   //asignación de código RGB para el color rojo
33.                     {255,255,0},  //asignación de código RGB para el color
    amarillo
34.                     {0,255,0}, //asignación de código RGB para el color verde
    limón
35.                     {0,255,255}, //asignación de código RGB para el color aqua
36.                     {0,0,255}, //asignación de código RGB para el color azul
37.                     {255,0,255} //asignación de código RGB para el color fucsia
38.                     };
39. //Asignación del parámetro alfa dependiendo de la profundidad
40.             float d=(depth-depthValueRange[0])*5.0f/(depthValueRange[1]-
    depthValueRange[0]);
41.             if(d<=0.0f)
42.                     {
43.                     for(int i=0;i<3;++i)
44.                         result[i]=GLubyte(mapColors[0][i]*0.2f);
45.                     }
46.             else if(d>=5.0f)
47.                     {
48.                     for(int i=0;i<3;++i)
49.                         result[i]=mapColors[5][i];
50.                     }
51.             else
52.                     {
53.                     int i0=int(d);
54.                     d-=float(i0);
55.                     for(int i=0;i<3;++i)
56.                         result[i]=GLubyte((mapColors[i0][i]*(1.0f-
    d)+mapColors[i0+1][i]*d)*(d*0.8f+0.2f));
57.                     }
58.             }
```

```
59.     }
```

5.4.4. Calibración

Para la calibración se hace el reconocimiento de un elemento de forma circular (figura 22) específicamente de 120 mm de diámetro de superficie blanca, comprobando si existe un elemento con estas características dentro de la superficie, una vez reconocido almacena el punto de enlace recién capturado y pasa a un nuevo punto, este proceso se repite entre 12 a 36 veces dependiendo de la calidad de calibración que se le quiera dar.

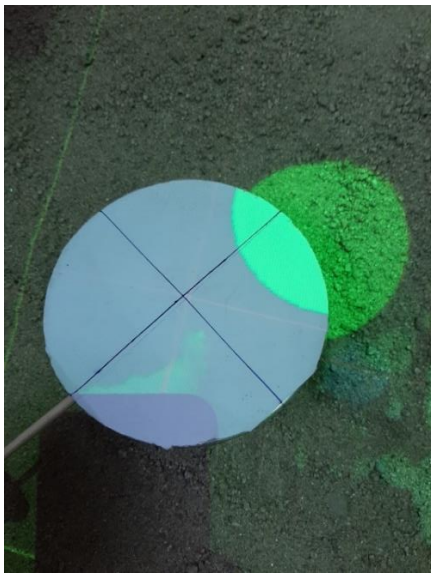

Figura 22: Elemento de Calibración
Fuente: Autor

```
1.  /*********************************
2.  CALIBRACIÓN DE AR SANDBOX
3.  *********************************/
4.  void CalibrateProjector::frame(void)
5.      {
6.  /* Comprobar si estamos capturando un punto de enlace y hay una nueva
    lista de discos extraídos:*/
7.      if(diskList.lockNewValue()&&capturingTiePoint&&diskList.getLocked
    Value().size()==1)
8.          {
9.  /* Accede al único disco extraído: */
10.         const Kinect::DiskExtractor::Disk& disk=diskList.getLockedValue
    ().front();
11.
12. /* Comprobar si existe una posición real en el centro del disco:*/
```

```
13.             bool diskValid=true;
14.             for(int i=0;i<3;++i)
15.                 diskValid=diskValid&&Math::isFinite(disk.center[i]);
16.
17.             #if 0
18.
19. /* Comprueba si el disco está dentro del área de sandbox:*/
20.             diskValid=diskValid&&(basePlane.getNormal()^(basePlaneCorner
    s[1]-basePlaneCorners[0]))*(disk.center-basePlaneCorners[0])>=0.0;
21.             diskValid=diskValid&&(basePlane.getNormal()^(basePlaneCorner
    s[3]-basePlaneCorners[1]))*(disk.center-basePlaneCorners[1])>=0.0;
22.             diskValid=diskValid&&(basePlane.getNormal()^(basePlaneCorner
    s[2]-basePlaneCorners[3]))*(disk.center-basePlaneCorners[3])>=0.0;
23.             diskValid=diskValid&&(basePlane.getNormal()^(basePlaneCorner
    s[0]-basePlaneCorners[2]))*(disk.center-basePlaneCorners[2])>=0.0;
24.             #endif
25.             if(diskValid)
26.                 {
27. /* Almacena el punto de enlace recién capturado: */
28.                 TiePoint tp;
29.                 int xIndex=tiePointIndex%numTiePoints[0];
30.                 int yIndex=(tiePointIndex/numTiePoints[0])%numTiePoints[1
    ];
31.                 int x=(xIndex+1)*imageSize[0]/(numTiePoints[0]+1);
32.                 int y=(yIndex+1)*imageSize[1]/(numTiePoints[1]+1);
33.                 tp.p=PPoint(Scalar(x)+Scalar(0.5),Scalar(y)+Scalar(0.5));
34.                 tp.o=disk.center;
35.                 tiePoints.push_back(tp);
36.
37.  /* Comprobar si es suficiente: */
38.                 --numCaptureFrames;
39.                 if(numCaptureFrames==0)
40.                     {
41.    /* Dejar de capturar este punto de enlace y pasar al siguiente: */
42.                     std::cout<<"done"<<std::endl;
43.                     capturingTiePoint=false;
44.                     ++tiePointIndex;
45.
46.   /* Comprobar si la calibración está completa: */
47.                     if(tiePointIndex>=numTiePoints[0]*numTiePoints[1])
48.                         {
49.   /* Calcular la transformación de calibración: */
50.                         calcCalibration();
51.                         }
52.                     }
53.                 }
54.             }
```

5.5. Selección de sistema operativo

Existe la posibilidad de implementar este simulador en dos sistemas operativos en concreto, Linux y Windows, los mismos que serán analizados para determinar cuál de los dos es más viable tomando en cuenta costos, proceso de instalación y la posibilidad de mejoras.

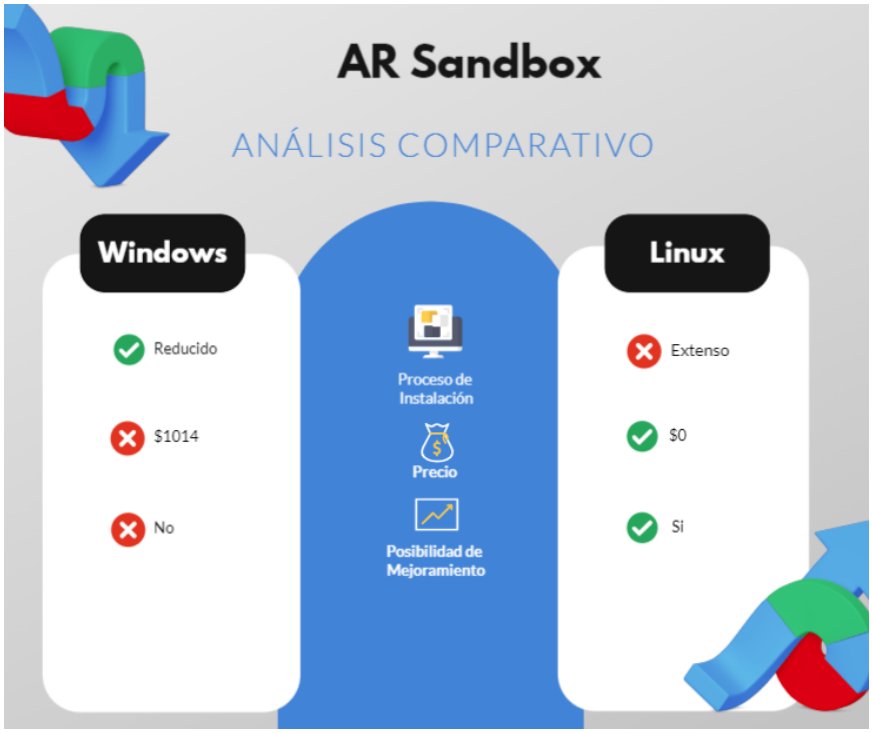

Figura 23: Análisis Comparativo
Fuente: Autor

5.5.1. Proceso de instalación

El proceso de instalación es uno de los parámetros a considerar para determinar la selección del sistema operativo. El software desarrollado para Windows es bastante intuitivo, provee una facilidad en el proceso de instalación y no hace falta conocimientos avanzados, esto gracias a que Windows está orientado al usuario y se puede ejecutar mediante un archivo ejecutable como se ve en la figura 24.

Figura 24: Archivo Ejecutable
Fuente: Autor

A diferencia del software desarrollado en Linux que tiene un nivel de complejidad mayor para la instalación debido a que todo este proceso se desarrolla mediante el uso de comandos en la terminal de Linux (figura 25), por lo mismo se debe tener conocimientos técnicos previos para poder realizarlo.

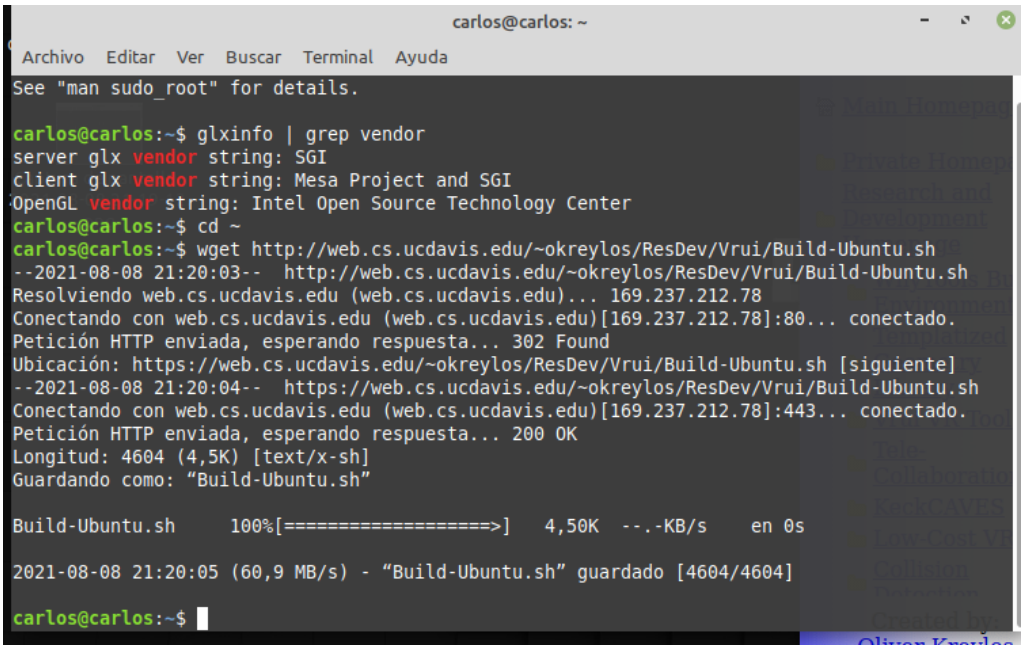

Figura 25: Terminal de linux
Fuente: Autor

5.5.2. Precio

El software desarrollado en Windows trabaja bajo una licencia para su uso (figura 26) la cual tiene un precio de aproximadamente 1014 dólares americanos, esta licencia es de por vida para una sola computadora.

Figura 26: Licencia de uso
Fuente: (AR-sandbox.eu, 2020)

Por el contrario, el software desarrollado en Linux es de libre acceso y no tiene ningún costo para su uso completo, basta con acceder al sitio web del desarrollador para encontrar toda la documentación del software (figura 27).

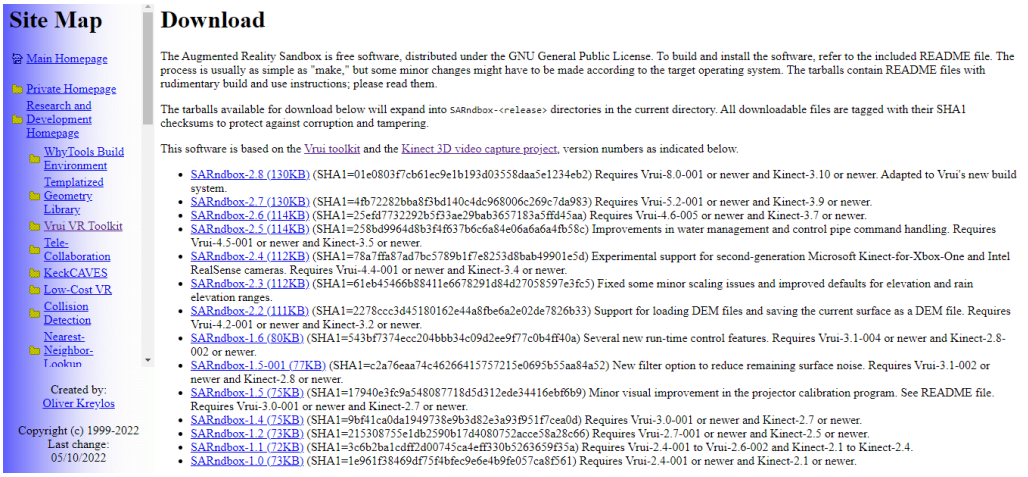

Figura 27: Sitio web del desarollador
Fuente: (UC Davis Department of Geology, 2022)

5.5.3. Posibilidad de mejoramiento

Con lo que respecta a la posibilidad de mejoramiento nos referimos a poder adicionar nuevas funciones y características a nuestro simulador. La posibilidad de mejoramiento en el software desarrollado en Windows es bastante baja a nivel de usuario, debido a que toda su documentación no es de acceso libre y no permite ningún cambio o modificación interna del software. Por el contrario, el software desarrollado en Linux se publica bajo el permiso al público en general (GPL) y por lo tanto se pueden realizar modificaciones de cualquier tipo siempre y cuando este no sea redistribuido.

6. Resultados

6.1. Instalación de los recursos de software

Para la instalación de los softwares necesarios para la implementación del simulador de mapa topográfico interactivo se seguirán los siguientes pasos para su correcta instalación, cabe recalcar que se debe seguir el mismo orden de instalación para la adaptación de los mismos.

Para el simulador de mapa topográfico se instaló las siguientes versiones:

- Linux Mint 19.3 "Tricia" - MATE (64 bits)
- Versión 8.0-001 de Vrui VR toolkit
- Versión 3.10 de Kinect 3D video Package
- Versión 2.8 de SARndbox

6.1.1. Instalar Linux

Instalar la versión de 64 bits de Linux Mint 19.3 con entorno de escritorio MATE, en una computadora de escritorio en blanco, es importante tomar en cuenta que no se debe utilizar una máquina virtual para su instalación debido a que AR Sandbox no funciona en la misma.

6.1.2. Instalar controladores de tarjeta gráfica

Instalar los controladores que proporciona el proveedor para su tarjeta gráfica, en este caso abrir el centro de control, seleccione “administrador de controladores”, luego seleccione el controlador binario recomendado y para finalizar seleccionar “aplicar cambios”.

Para finalizar este paso se debe comprobar que el controlador está funcionando correctamente, reiniciando la computadora y abriendo luego una ventana del terminal e ingresando el siguiente comando.

glxinfo | grep vendor

La terminal debe responder con lo siguiente:

server glx vendor string: NVIDIA Corporation
client glx vendor string: NVIDIA Corporation
OpenGL vendor string: NVIDIA Corporation

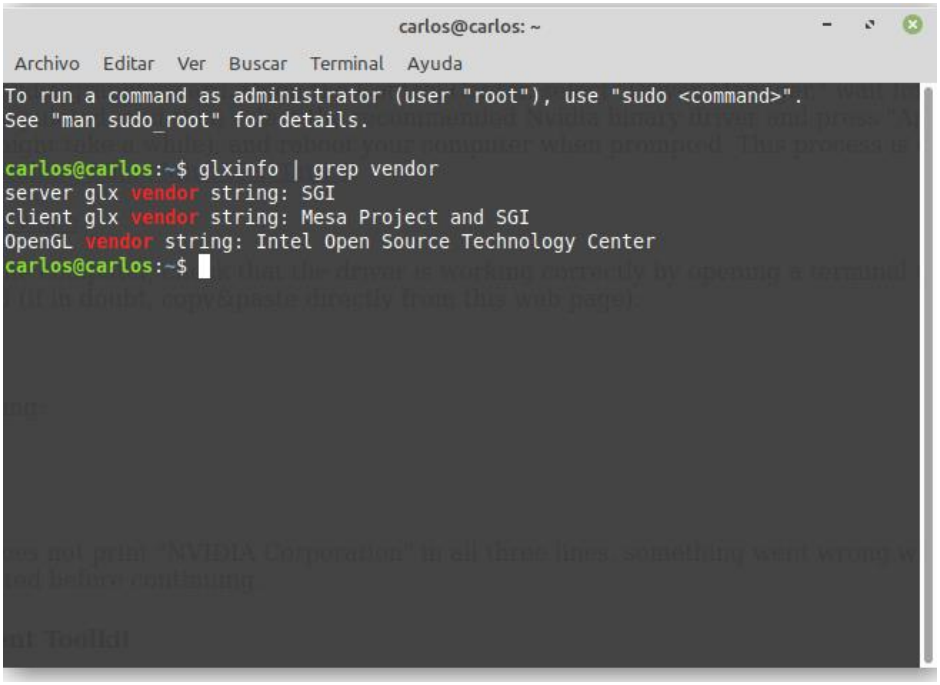

Figura 28:Verificación de controladores de tarjeta gráfica
Fuente: Autor

6.1.3. Instalar el kit de herramientas de desarrollo de Vrui VR

Abra una ventana de terminal e ingresar el siguiente comando.

cd ~
wget http://web.cs.ucdavis.edu/~okreylos/ResDev/Vrui/Build-Ubuntu.sh

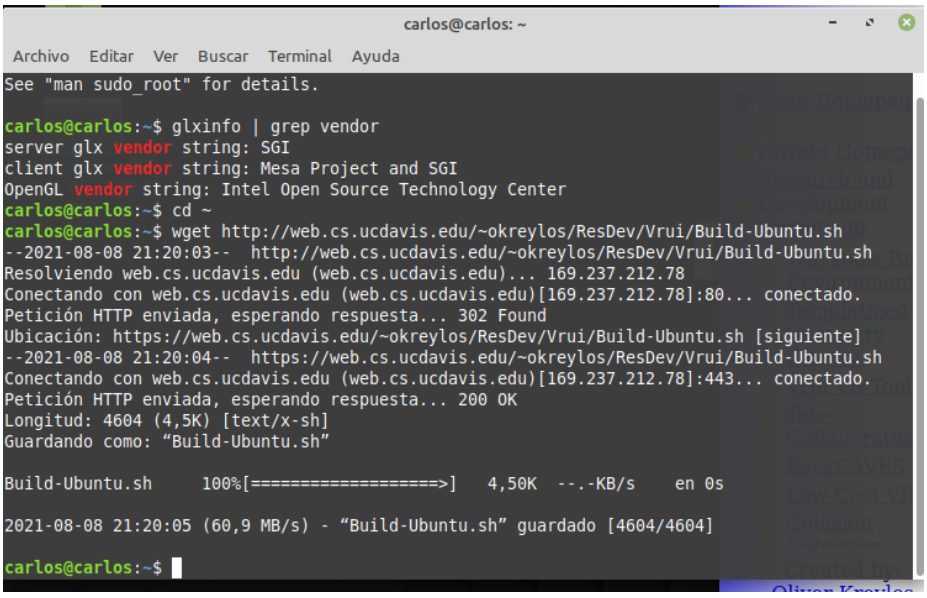

Figura 29: Instalación de kit de herramientas de Vrui VR
Fuente: Autor

bash Build-Ubuntu.sh

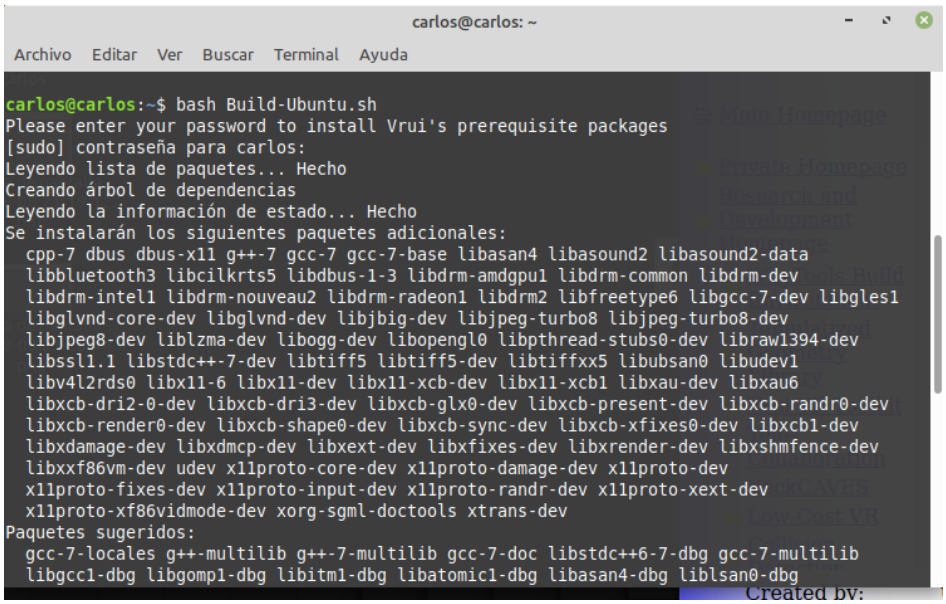

Figura 30: Instalación de kit de herramientas de Vrui VR
Fuente: Autor

El último comando le pedirá que ingrese la contraseña de su usuario para instalar las bibliotecas de requisitos previos, esto podrá llevar un poco de tiempo y producirá una gran cantidad de resultados. Al finalizar este proceso se mostrará una ventana con una imagen de un globo terráqueo giratorio. Podrá manipular este entorno para adaptarse un poco a la manipulación del mismo de la siguiente manera.

Figura 31: Ventana con globo terráqueo giratorio
Fuente: Autor

- **Menú Principal:** mantener presionado el botón derecho del mouse para que el menú principal de la aplicación se muestre. Luego mueva el mouse a cualquier lado del menú y suelte el botón derecho del mouse.
- **Rotar:** mantener presionado el botón izquierdo del mouse, al mover el mouse hará que la vista 3D rote en la ventana.
- **Mover:** para mover el globo hacia la izquierda, derecha, arriba, abajo en la ventana, mantenga presionado la tecla "z" y mueva el mouse hacia la izquierda, derecha, arriba o abajo.
- **Escala:** Para encoger o hacer crecer el bloque en la ventana y mantenga presionadas la tecla "z" y el botón izquierdo del mouse, y mueva el mouse hacia arriba o hacia abajo.
- **Volver a centrar:** para volver a centrar la vista del globo terráqueo en la ventana mantenga presionada conjuntamente la tecla "Windows" y luego presione la tecla la tecla "Inicio".

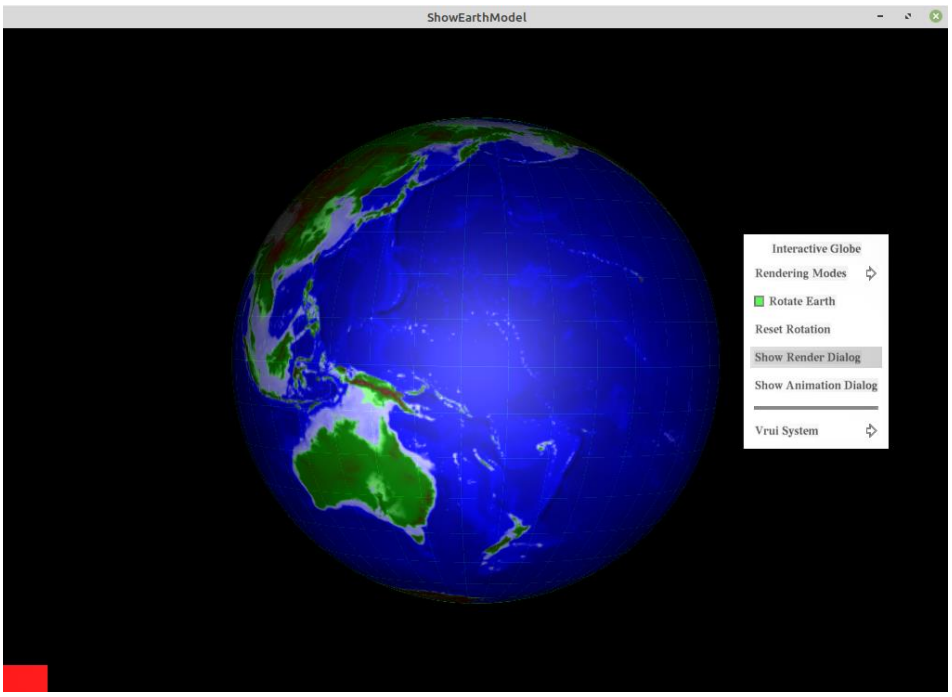

Figura 32: Menú principal de la Ventana con globo terráqueo giratorio
Fuente: Autor

Cuando se haya terminado de explorar la venta del globo terráqueo, cierre la ventana y vuelva a la ventana del terminal.

6.1.4. Instalar el paquete de video Kinect 3D

Para instalar el paquete de video Kinect 3D inserte los siguientes comandos en la ventana del terminal.

cd ~/src
wget http://web.cs.ucdavis.edu/~okreylos/ResDev/Kinect/Kinect-3.10.tar.gz

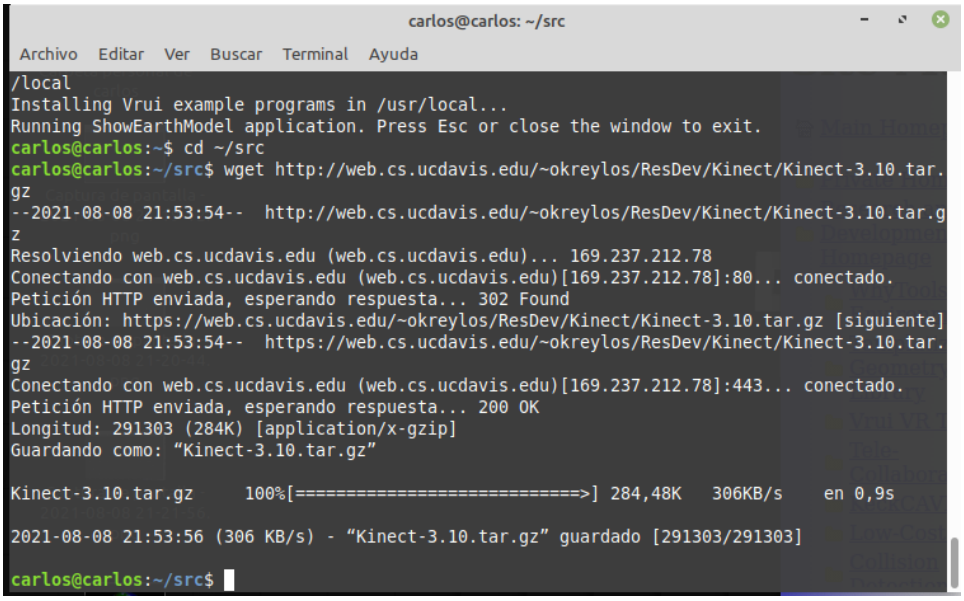

Figura 33: Instalación de paquete de video Kinect 3D
Fuente: Autor

tar xfz Kinect-3.10.tar.gz
cd Kinect-3.10
make

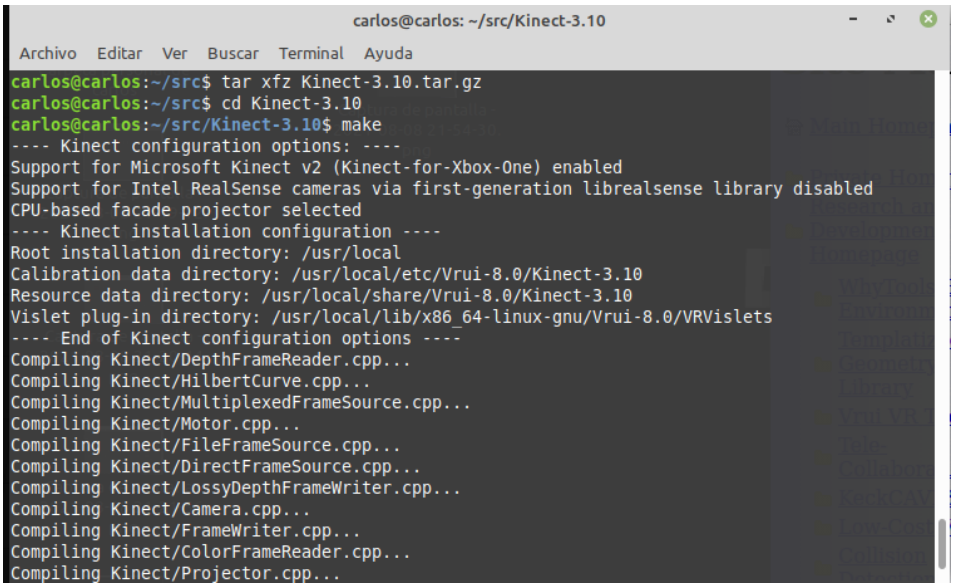

Figura 34: Instalación de paquete de video Kinect 3D
Fuente: Autor

sudo make install
sudo make installudevrules
ls /usr/local/bin

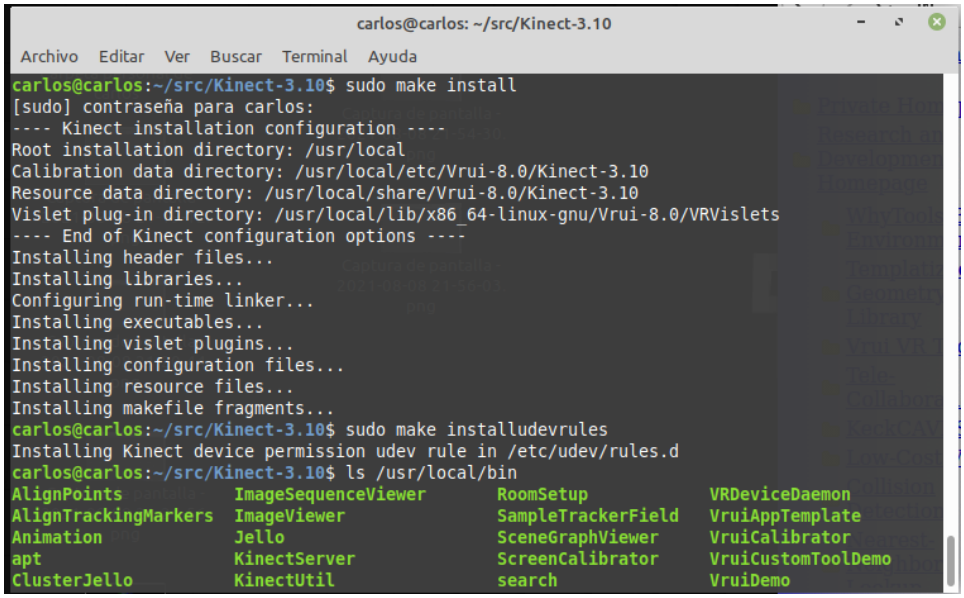

Figura 35: Instalación de paquete de video Kinect 3D
Fuente: Autor

Si en alguno de los comandos le solicite que ingrese la contraseña, hágalo sin ningún problema para poder continuar. Al finalizar compruebe que la lista de nombres impresa por el ultimo comando contenga KinectUtil y RawKinectViewer.

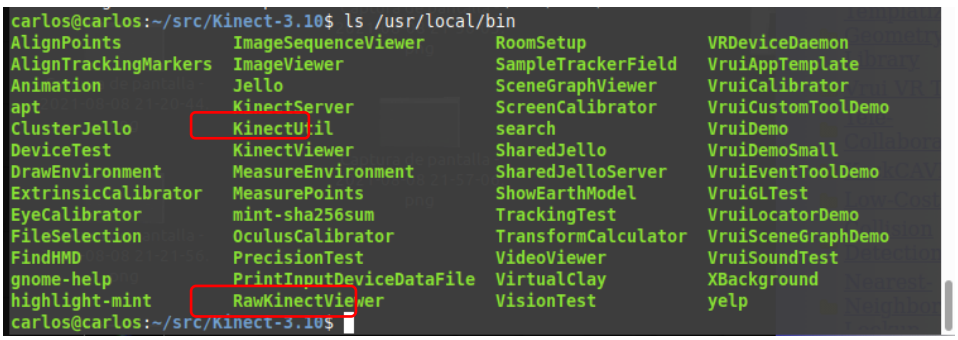

Figura 36: Comprobación de la lista impresa
Fuente: Autor

6.1.5. Instalar el Sandbox de realidad aumentada

Para instalar el software Sandbox de realidad aumentada ejecute los siguientes comandos en la misma ventana de la terminal.

cd ~/src
wget http://web.cs.ucdavis.edu/~okreylos/ResDev/SARndbox/SARndbox-2.8.tar.gz

tar xfz SARndbox-2.8.tar.gz

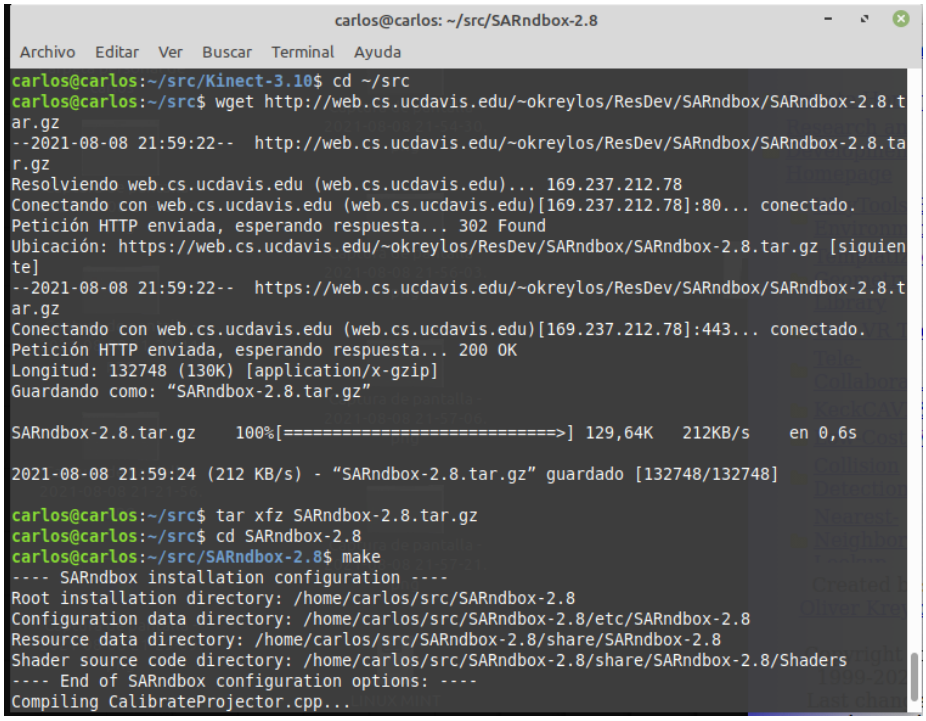

Figura 37: Instalación de Sandbox de realidad aumentada
Fuente: Autor

cd SARndbox-2.8
make
ls ./bin

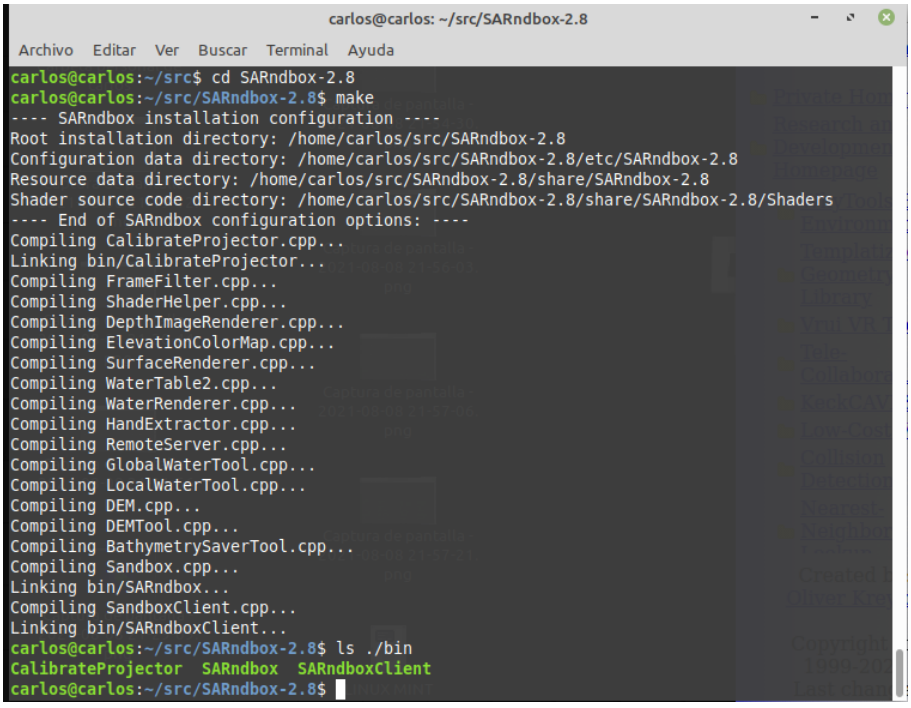

Figura 38: Instalación de Sandbox de realidad aumentada
Fuente: Autor

comprobar que la lista de nombres impresa por el ultimo comando contenga CalibrateProjector, SARndbox y SARndboxClient.

Figura 39: Comprobación de la lista impresa
Fuente: Autor

6.1.6. Conectar y configurar la cámara 3D

Se conecta la cámara 3D y ejecute el siguiente comando en la terminal para descargar los parámetros de calibración intrínsecos.

sudo /usr/local/bin/KinectUtil getCalib 0

6.1.7. Alinear Kinect sobre la zona de pruebas

Para poder alinear el Kinect, se coloca la cámara Kinect de manera que mire a la superficie de pruebas y luego ejecute el siguiente comando en la terminal.

sudo /usr/local/bin/RawKinectViewer -compress 0

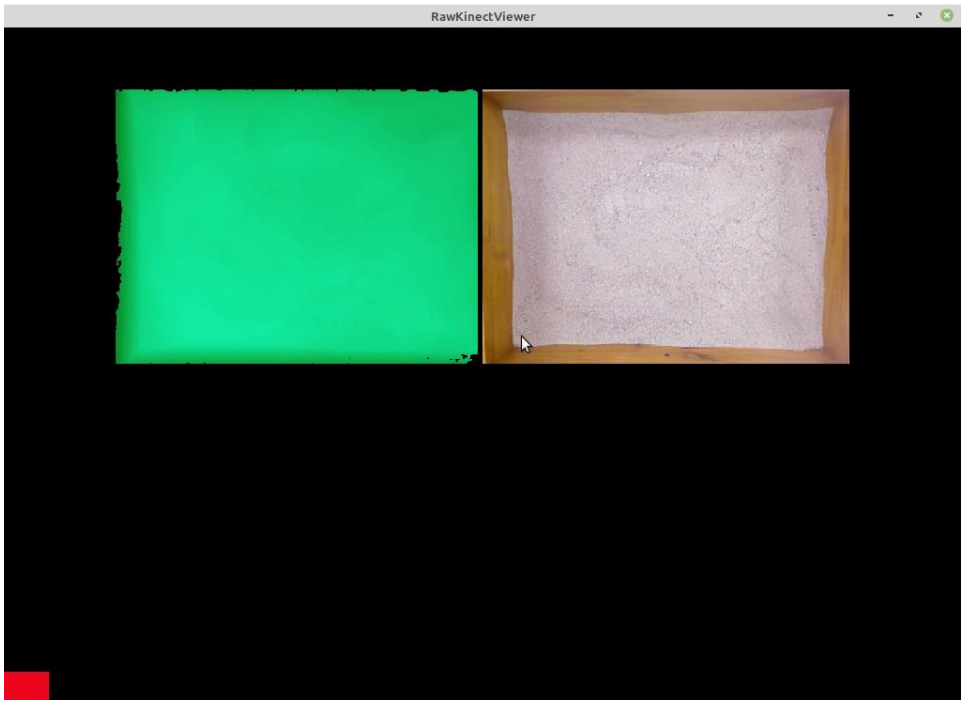

Figura 40: Alineación de Kinect sobre la zona de pruebas
Fuente: Autor

6.1.8. Medir la ecuación de plano base de Sandbox

Luego de haber llenado la caja con arena se calcula el plano base colocando un pedazo de cartulina encima de la caja de arena.

Desde la terminal ejecuta el comando:

RawKinectViewer -compress 0

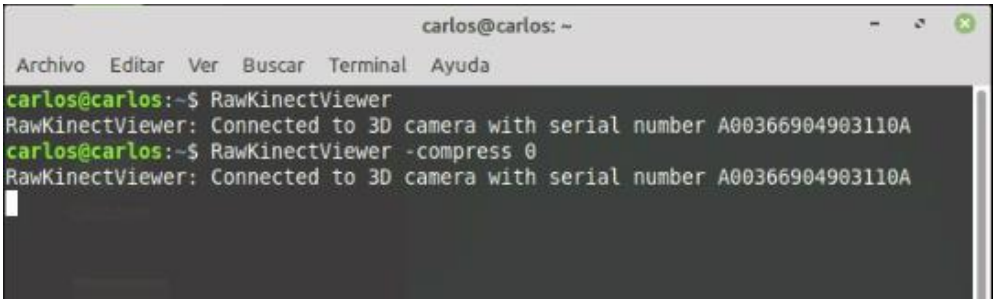

Figura 41: Comando para medir ecuación de plano base
Fuente: Autor

Mantener presionado el botón derecho del mouse, mover el cursor sobre la opción Average Frames y luego soltar el botón del mouse.

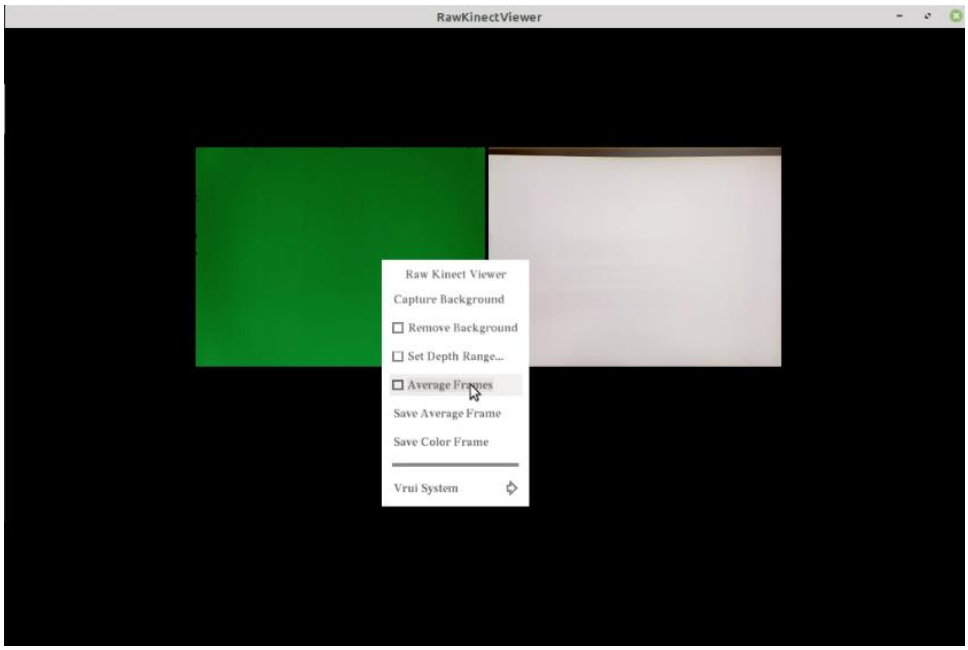

Figura 42: Lista de opciones de configuración de Kinect
Fuente: Autor

Presionar la tecla 1, mueva el cursor sobre la opción Extract Planes y luego suelte la tecla 1.

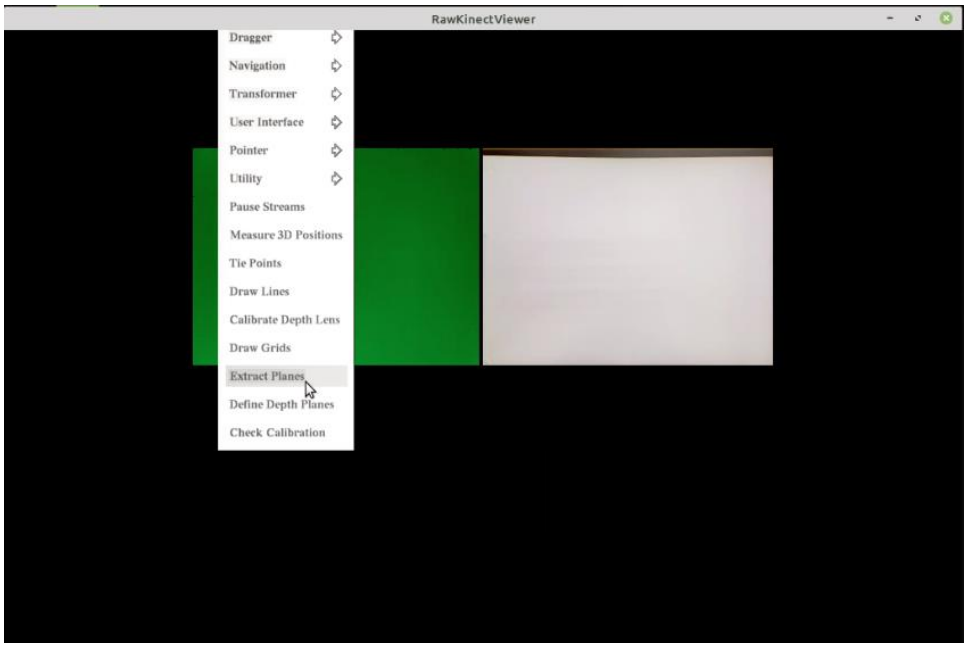

Figura 43: lista de opciones para extraer planos
Fuente: Autor

Dibujar un rectángulo que se ajuste al interior de su superficie plana, se sugiere dejar espacio entre el rectángulo y los bordes de la superficie plana.

Ubicar el cursor en la esquina superior izquierda de la superficie plana, manteniendo presionada la tecla 1 y arrastrar el mouse formando un rectángulo hacia la esquina inferior derecha, luego suelte la tecla 1.

Figura 44: ajuste de la vista de profundidad
Fuente: Autor

Presione la tecla Esc para cerrar la ventana de RawKinectViewer

En la terminal se mostrará dos líneas, seleccionar y copiar la parte de la segunda línea que se muestra a continuación.

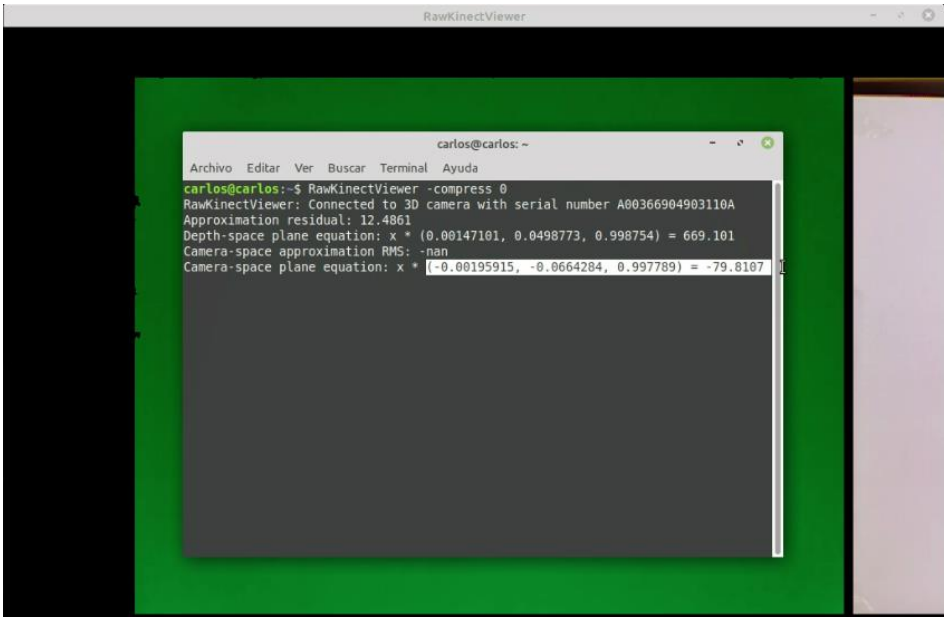

Figura 45: Código de respuesta a la extracción del plano
Fuente: Autor

Editar el BoxLayout.txt ejecutando el siguiente comando en la terminal.

cd ~/src/SARndbox-2.8
xed etc/SARndbox-2.8/BoxLayout.txt &

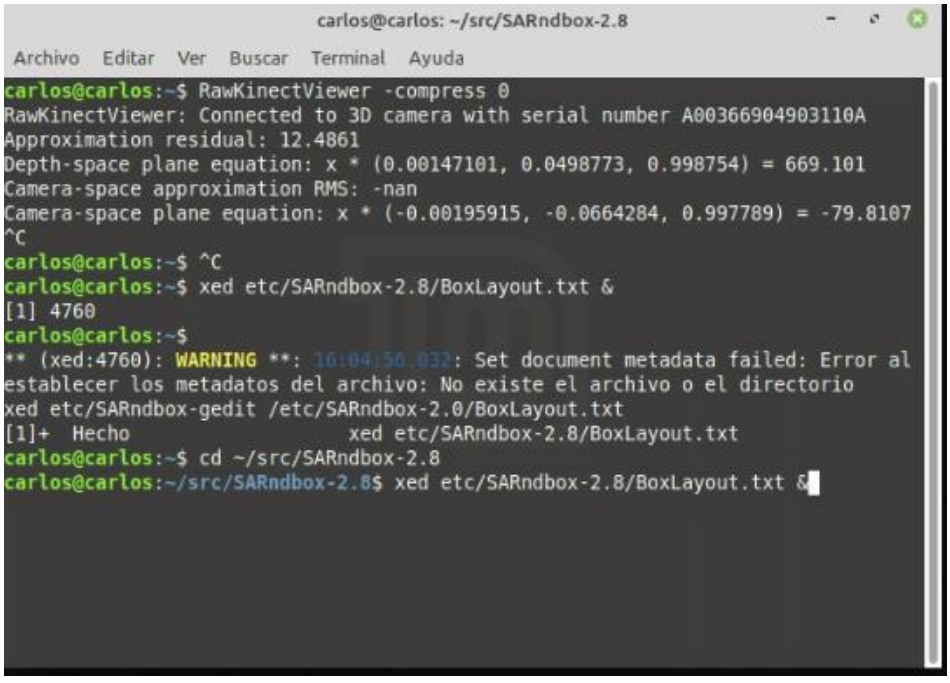

Figura 46: Comando para abrir archivo BoxLayot.txt
Fuente: Autor

Una vez insertado el comando se abrirá la siguiente ventana.

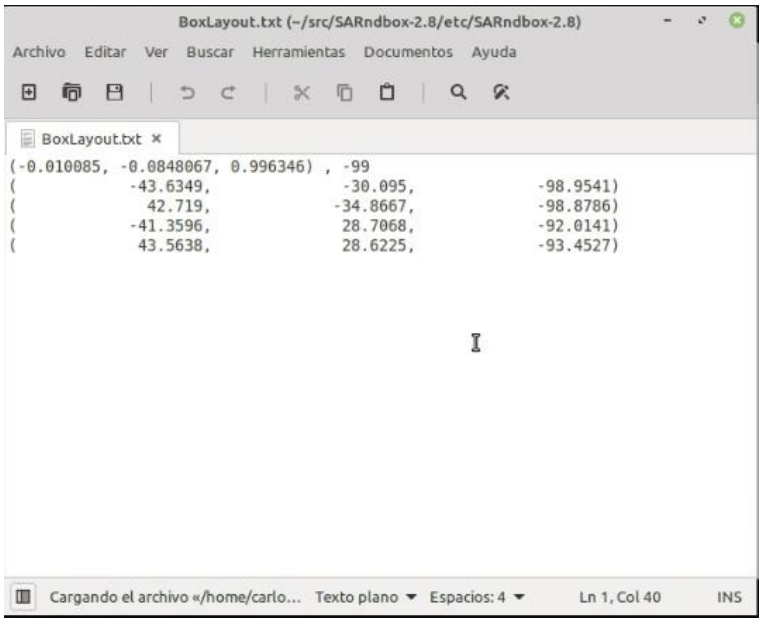

Figura 47: Ventana de archivo BoxLayout.txt
Fuente: Autor

Pegar la línea que acaba de copiar sobre la primera línea del archivo BoxLayout.txt, así mismo remplace el signo "=" por una ",".

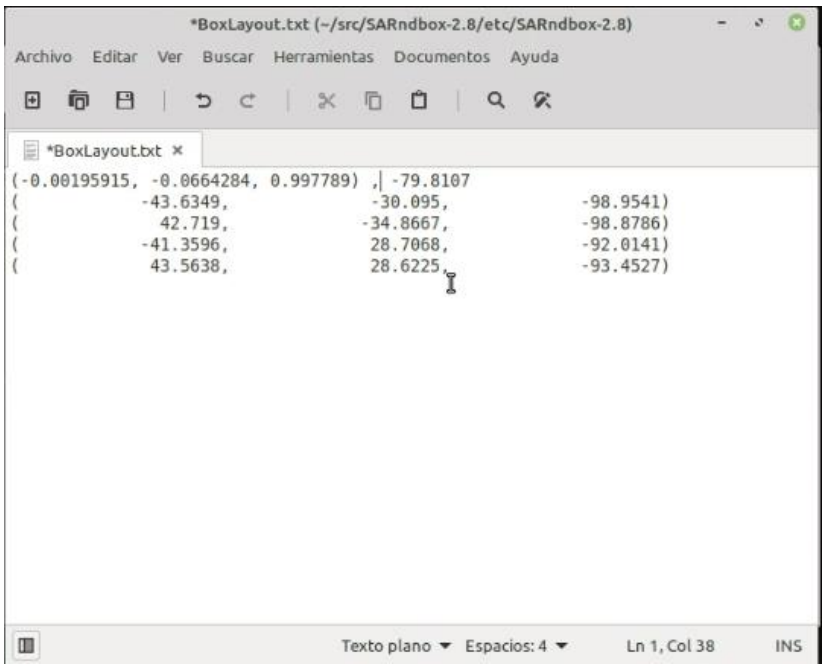

Figura 48: Ventana de archivo BoxLayout.txt
Fuente: Autor

Culminado esto guardar y cerrar la ventana del archivo BoxLayout.txt

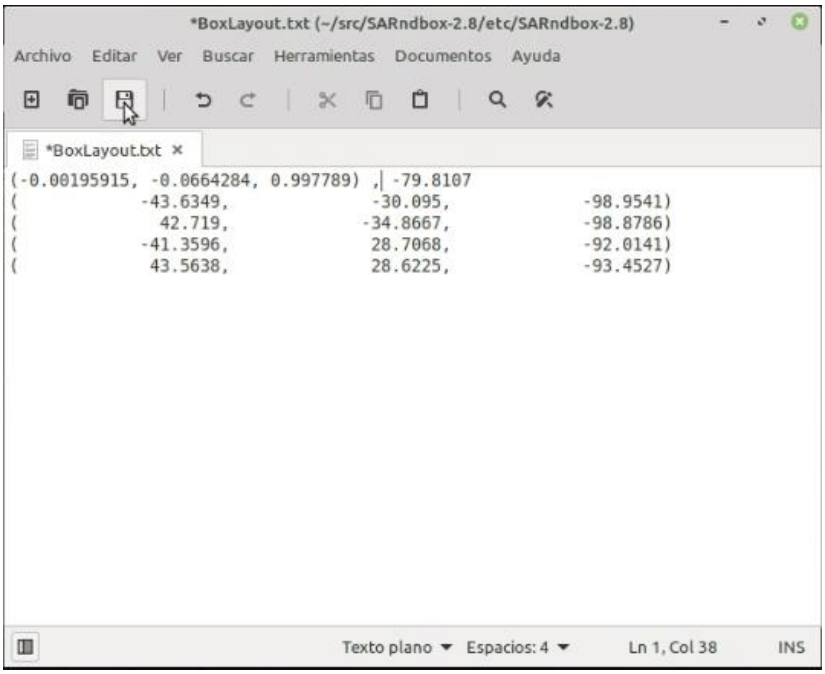

Figura 49: ventana de archivo BoxLayout.txt
Fuente: Autor

6.1.9. Medir las posiciones de las esquinas de la caja 3D de Sandbox

Una vez llenada la caja de arena, haciendo que la superficie de la arena este lo más nivelada posible.

Desde la terminal ejecutar la siguiente línea de comando.

RawKinectViewer -compress 0

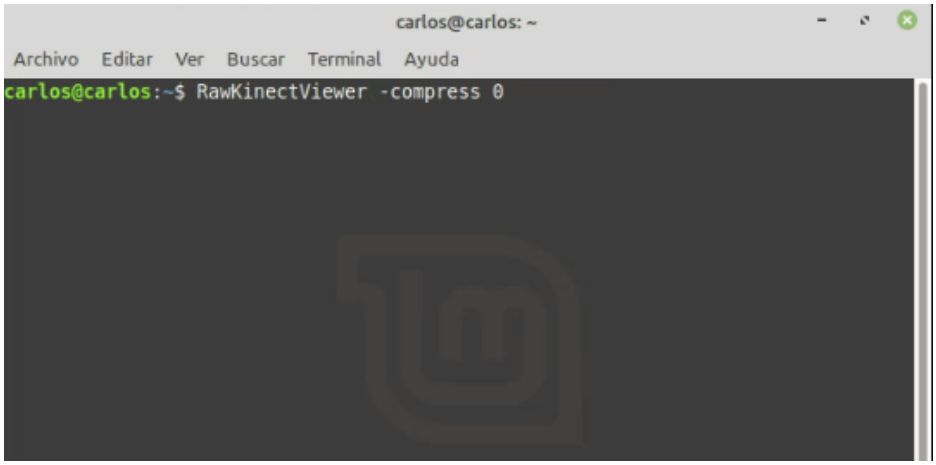

Figura 50: Código para medir la posición de las esquinas de la caja
Fuente: Autor

Mantener presionado el botón derecho del mouse y mover el cursor sobre la opción de Average Frames y soltar el botón del mouse.

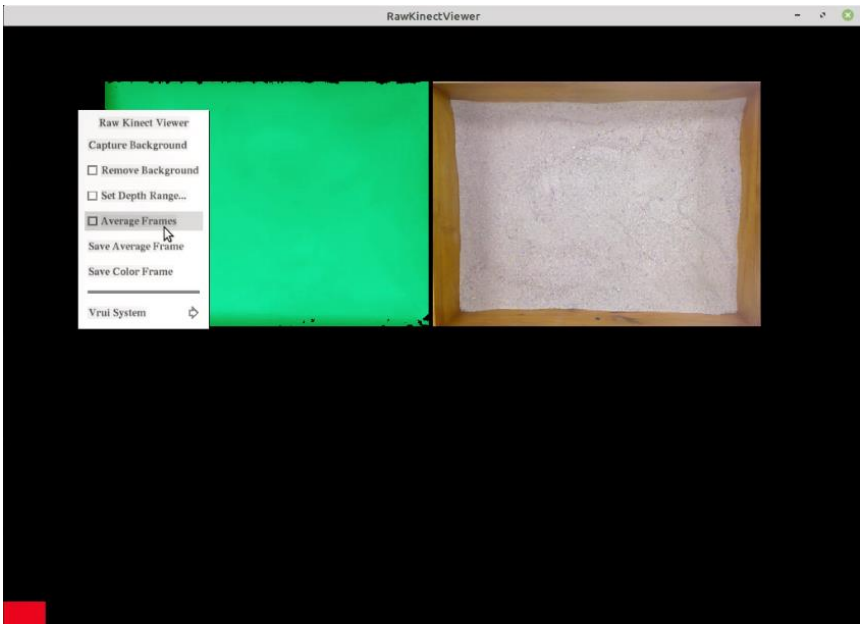

Figura 51: Lista de opciones para la configuración del Kinect
Fuente: Autor

Mantener presionado la tecla 1, deslizar el cursor sobre la opción Measure 3D Positions y luego suelte la tecla 1.

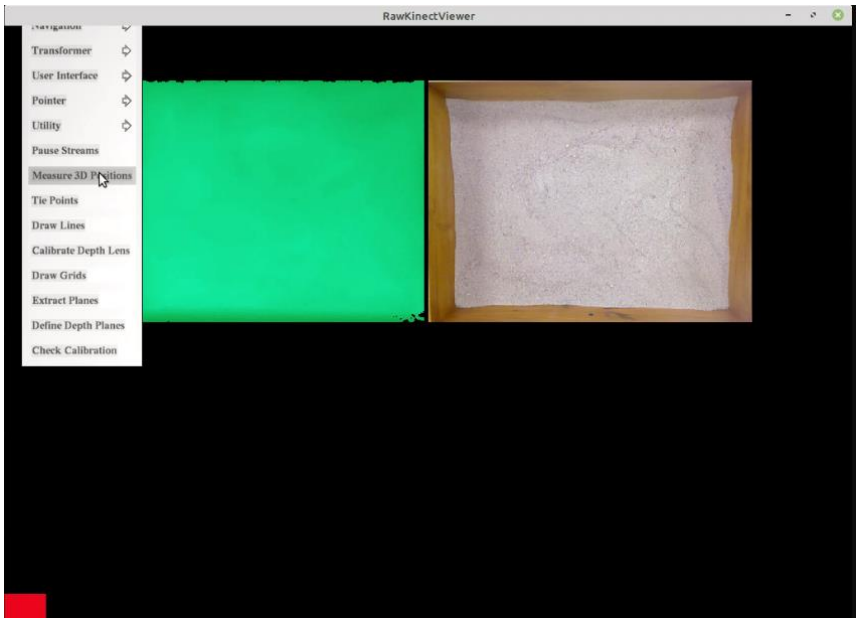

Figura 52: lista de opciones para medir posiciones 3D
Fuente: Autor

Mover el cursor a la esquina inferior izquierda y presione la tecla 1.

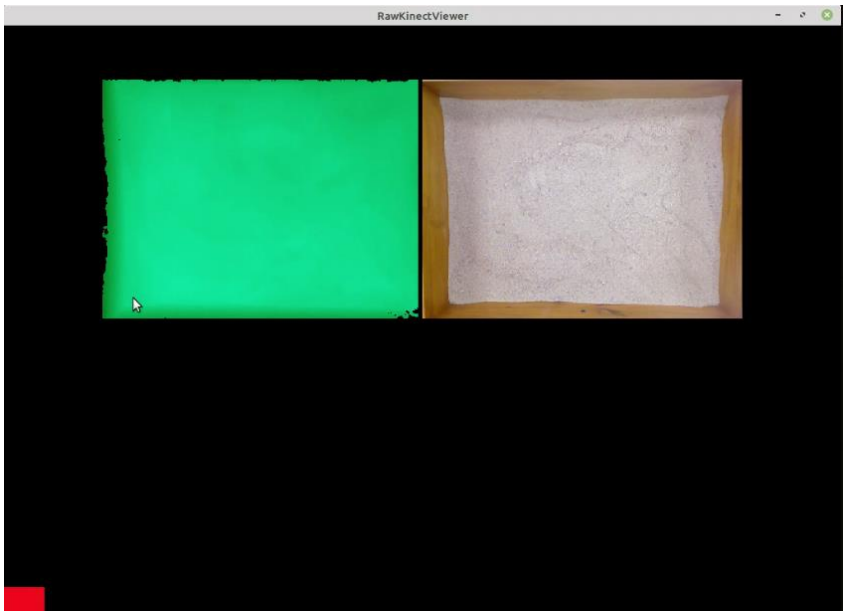

Figura 53: selección de la esquina inferior izquierda de la caja
Fuente: Autor

Luego mover el cursor a la esquina inferior derecha y presione la tecla 1.

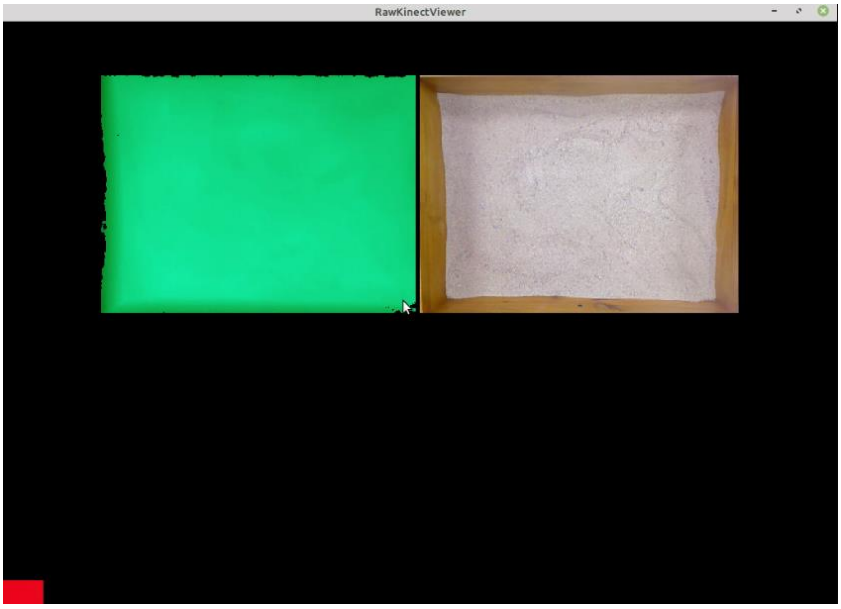

Figura 54: selección de la esquina inferior derecha de la caja
Fuente: Autor

Seguidamente mover el cursor a la esquina superior izquierda y presione la tecla 1.

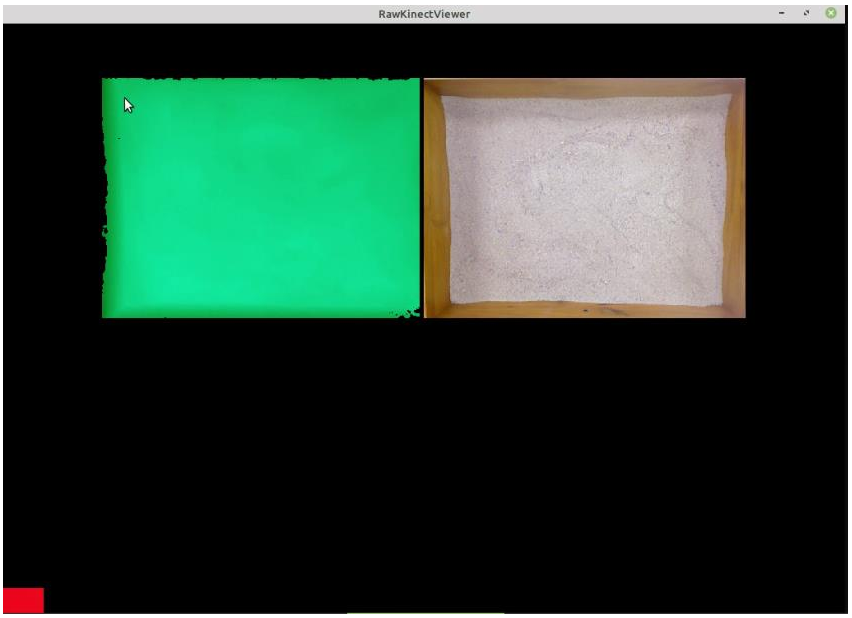

Figura 55: selección de la esquina superior izquierda de la caja
Fuente: Autor

Por último, mover el cursor a la esquina superior derecha y presionar la tecla 1.

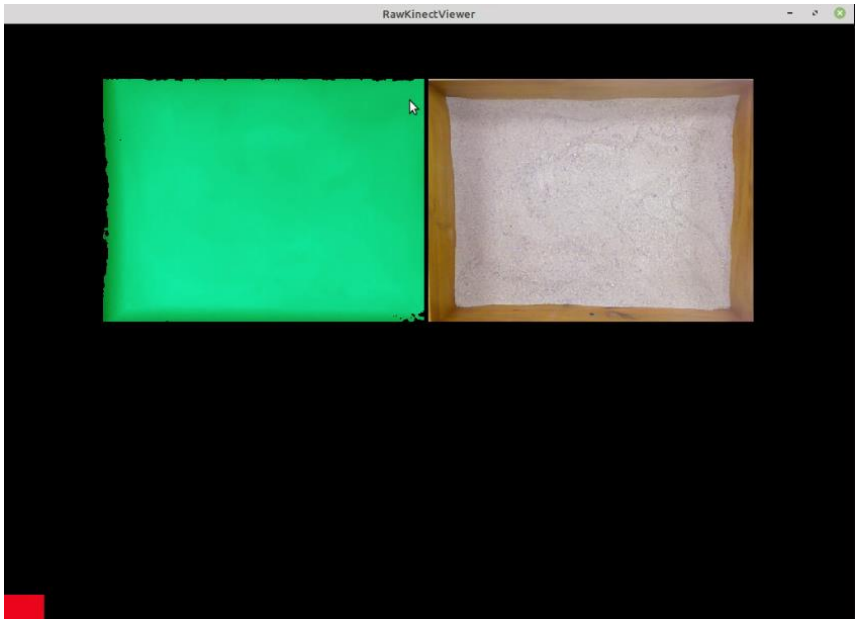

Figura 56: selección de la esquina superior derecha de la caja
Fuente: Autor

Luego de esto presionar Esc para salir de la ventana de RawKinectViewer y se podrá ver la ventana de la terminal.

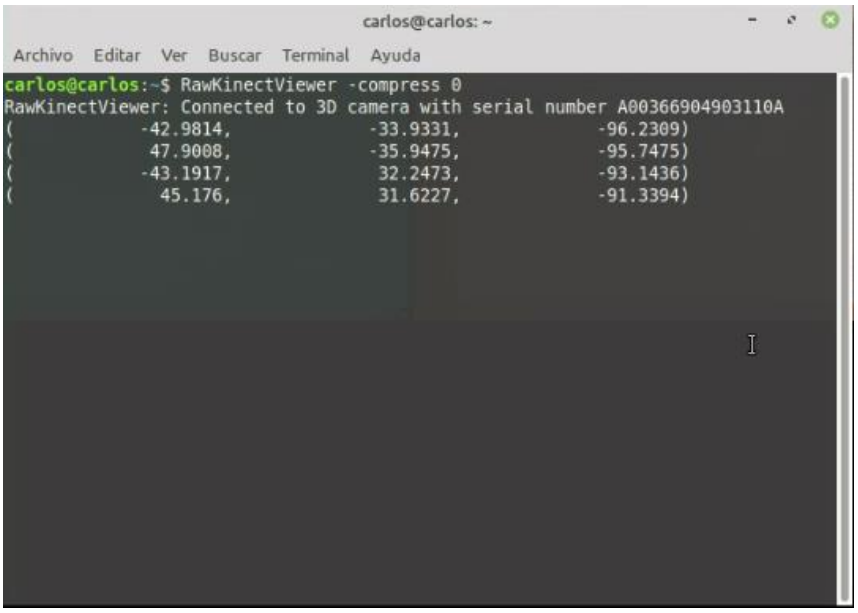

Figura 57: Código del sistema para medir la posición de las extensiones 3D
Fuente: Autor

En la ventana de la terminal se mostrará cuatro líneas, las cuales debe seleccionar y copiar.

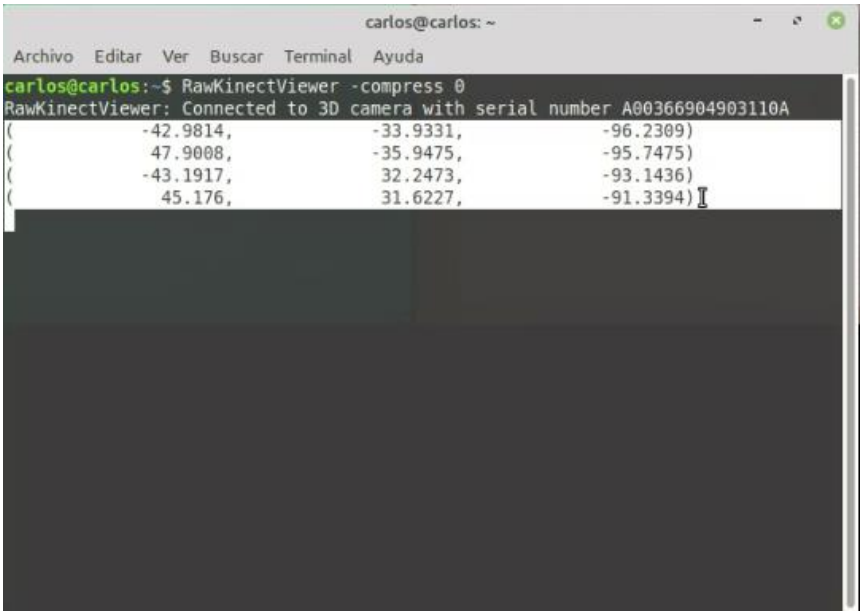

Figura 58: código del sistema para medir la posición de las extensiones 3D
Fuente: Autor

Editar el archivo BoxLayout.txt ejecutando el siguiente comando en la terminal.

xed etc/SARndbox-2.8/BoxLayout.txt &

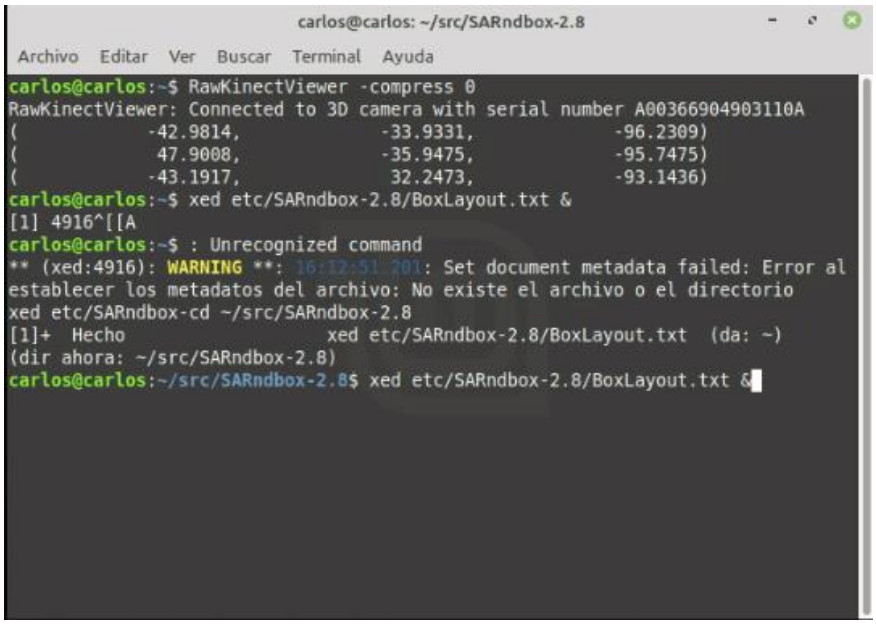

Figura 59: Comando para editar archivo BoxLayout.txt
Fuente: Autor

Este mostrara la ventana del archivo BoxLayout.txt.

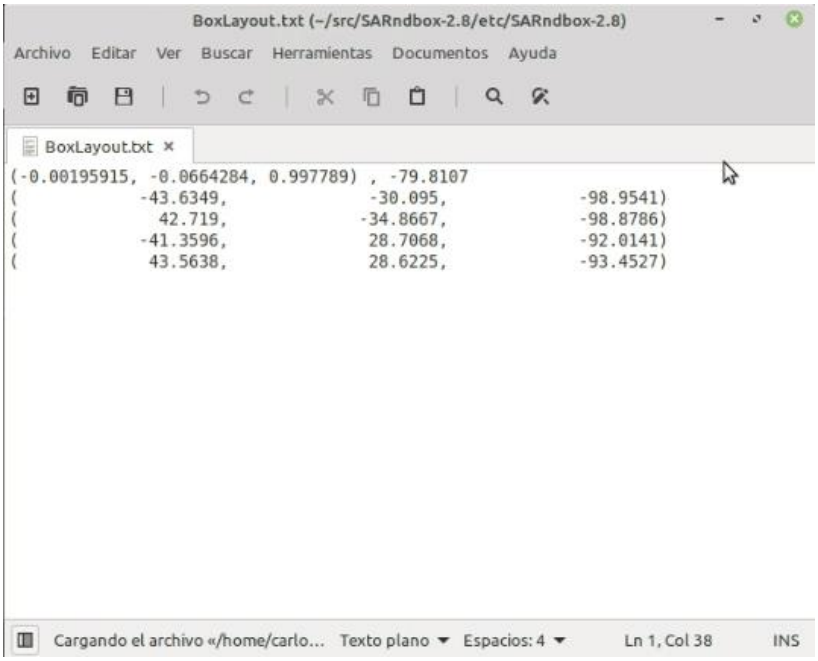

Figura 60: Ventana de archivo BoxLayout.txt
Fuente: Autor

Remplazar las cuatro últimas líneas del archivo BoxLayout.txt con las cuatro líneas que copio con anterioridad.

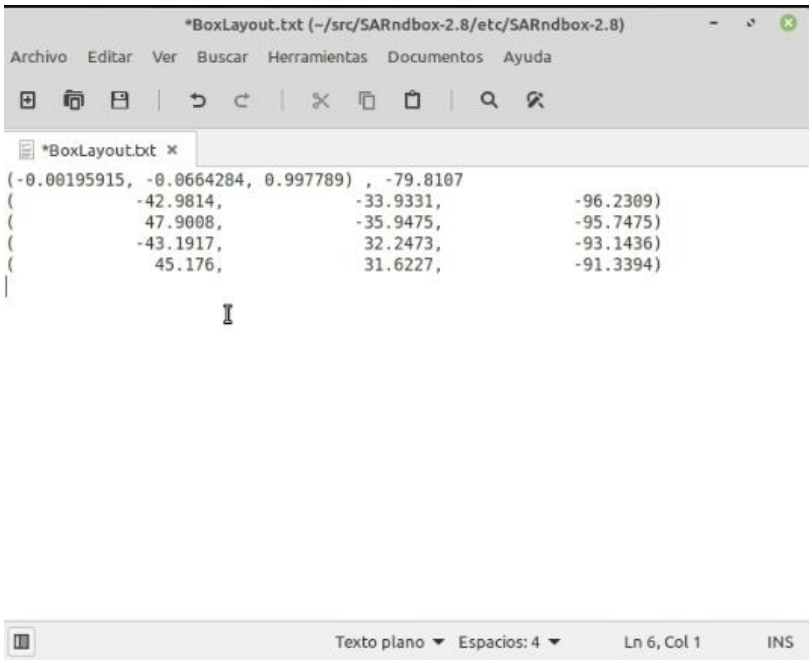

Figura 61: Ventana de archivo BoxLayout.txt
Fuente: Autor

Guarde el archivo y cierre la ventana del BoxLayout.txt

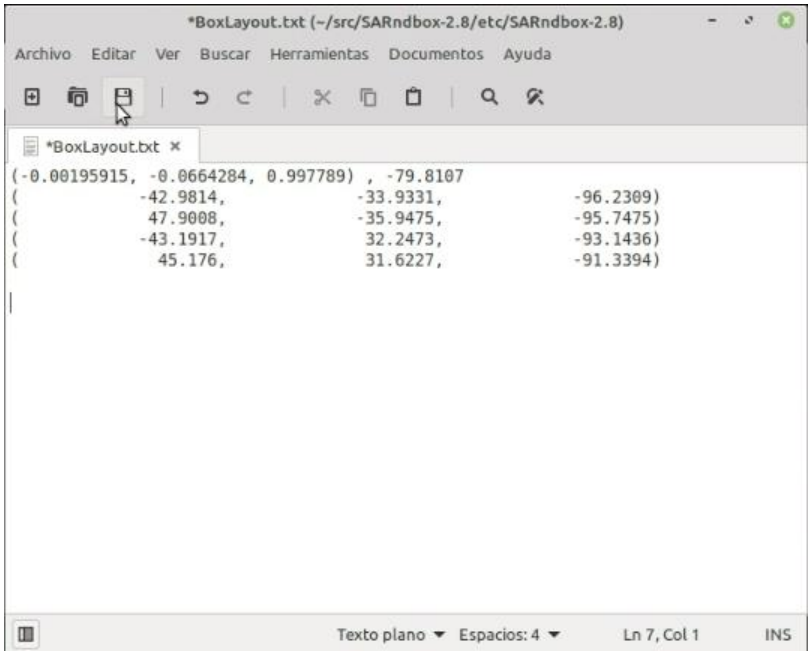

Figura 62: Ventana de archivo BoxLayout.txt
Fuente: Autor

6.1.10. Proyección de posición

Encender el proyector y conectarlo a la computadora. Configurar la pantalla y la proyección de la misma accediendo al menú de búsqueda la cual presentara la siguiente ventana.

Seleccionar la sección de pantallas y configurar la resolución y la frecuencia de actualización.

Escribir en la terminal el siguiente comando para abrir la ventana XBackground para ajustar la proyección sobre la superficie.

XBackground

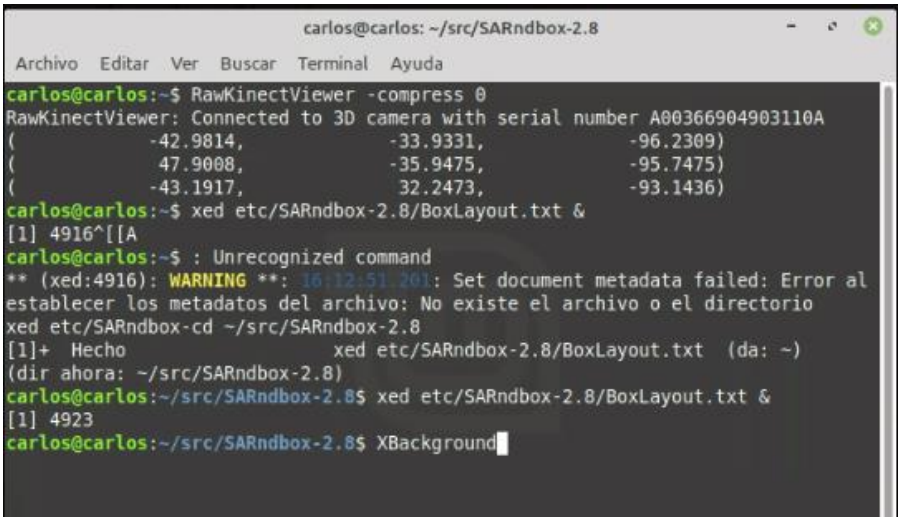

Figura 63: Comando para ajuste del área de la caja de arena
Fuente: Autor

Con la proyección de esta ventana ajustar la imagen de manera que llene el interior de la caja de arena.

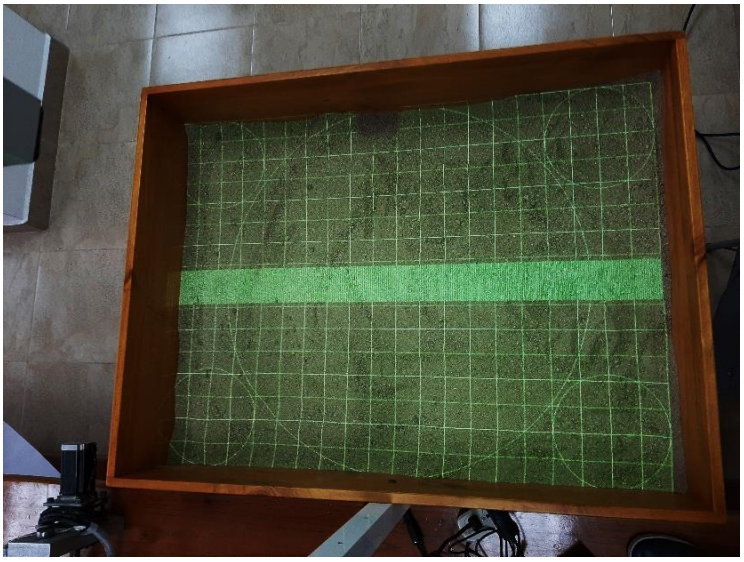

Figura 64: Proyección de la ventana de ajuste Xbachground
Fuente: Autor

Al finalizar el ajuste del proyector, presionar Esc para cerrar la ventana XBackground.

6.1.11. Calibrar AR Sandbox

Para la calibración de la caja de arena se usará un objeto de alineación, en este caso un CD (figura 65) para realizar la calibración a varias alturas.

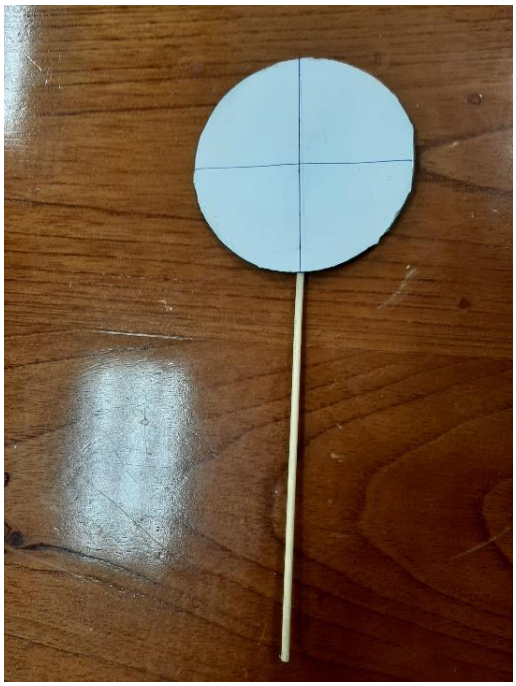

Figura 65: Objeto de Alineación
Fuente: Autor

En cada altura se capturarán 12 puntos de la superficie, pero para su calibración optima es recomendable capturar puntos de unión a tres alturas diferentes, es decir se tendrá un total de 36 puntos.

Para esto se ejecutará el siguiente comando en la terminal.

cd ~/src/SARndbox-2.8
./bin/CalibrateProjector -s <width> <height>

Donde <width> y <height> son el ancho y alto de la imagen de su proyector en pixeles, respectivamente. Estos datos los puede encontrar en la configuración de la pantalla que se realizó en el paso 10.

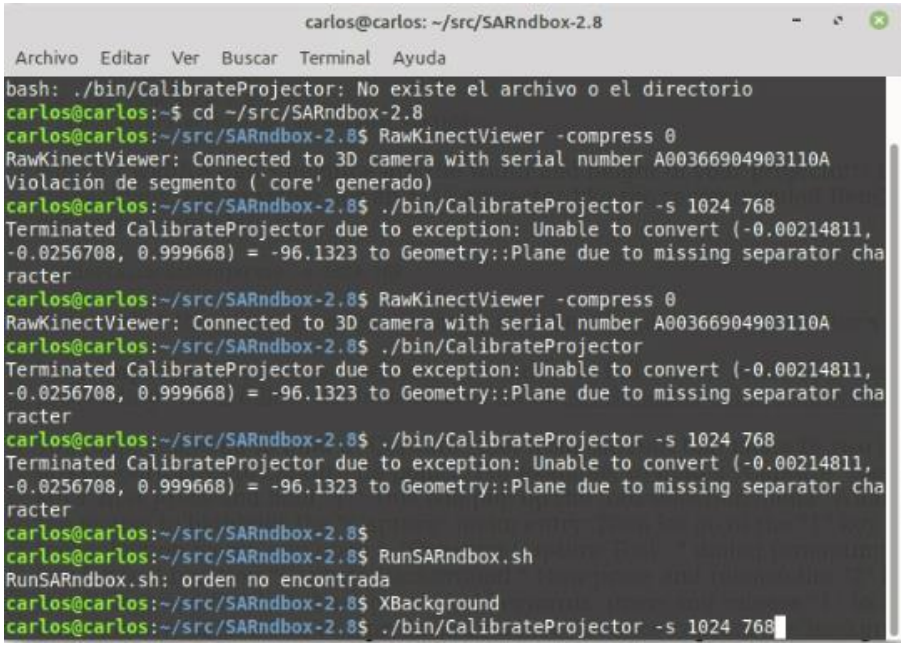

Figura 66: Comando para el ajuste del ancho y el alto de la imagen
Fuente: Autor

A continuación, se captura los puntos de unión a la altura más baja, mediante el espaciador más corto alinee el objeto (CD) debajo de la cruz que se proyecta en la superficie de la arena.

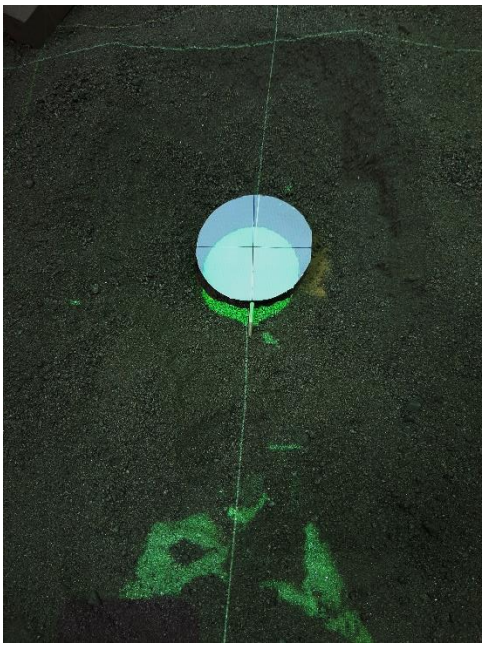

Figura 67: Captura de punto de unión
Fuente: Autor

Luego presionar la tecla 1 para capturar el punto, seguidamente el software mostrará el siguiente punto donde tendrá que ubicar el CD. Repita este proceso para los 11 puntos restantes a esta profundidad.

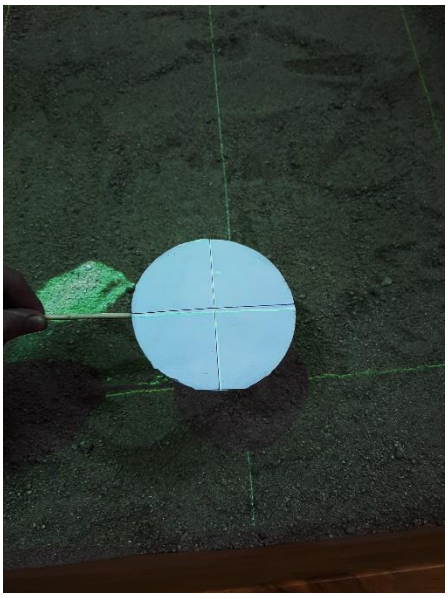

Figura 68: Captura de puntos de unión
Fuente: Autor

Una vez que la cruz haya regresado a la posición donde se inició, se deberá cambiar el espaciador por uno más alto. Este proceso se repite para los 2 espaciadores restantes.

Sin embargo, esto no es una regla que se debe seguir, debido a que existen varias formas de realizar este proceso, una de ellas es alternar los espaciadores de diferente nivel dentro de los primeros 11 puntos.

Al terminar este paso se podrá visualizar la proyección de dos líneas rojas que se intersecan, las cuales se alinean con el objeto de alineación.

6.1.12. Ejecución AR Sandbox

Una vez finalizado el proceso de instalación, configuración y calibración ejecutar en el siguiente comando en la terminal para iniciar la aplicación.

RunSARndbox.sh

Con este comando se iniciará el programa y se podrá visualizar las proyecciones de los relieves en la caja de arena, representado mediante colores. A partir de este punto ya se podrá modificar la superficie de la arena formando montañas o relieves al gusto del usuario.

Figura 69: Proyección de relieve
Fuente: Autor

6.1.13. Crear icono en el escritorio para abrir AR Sandbox

Iniciar AR Sandbox puede requerir se inserten varias líneas de comando en la ventana de la terminal, lo que puede resultar algo molesto y tedioso. Para simplificar este proceso se puede crear un icono en el escritorio para iniciar AR Sandbox simplemente haciendo doble clic sobre el icono.

Primero se debe crear un script de Shell para iniciar la aplicación, esta debe contener todos los argumentos de la línea de comandos. Este script se crea ejecutando en una ventana de la terminal el siguiente comando.

xed ~/src/SARndbox-2.8/RunSARndbox.sh

Figura 70: Comando para crear el script

Fuente: Autor

Esto abrirá una ventana del editor con un archivo vacío, en la que debe pegar exactamente las siguientes líneas. Lo que hará esto será abrir el directorio donde se encuentra el ejecutable de la aplicación.

```
#!/bin/bash

# Enter SARndbox directory:
cd ~/src/SARndbox-2.8

# Run SARndbox with proper command line arguments:
./bin/SARndbox -uhm -fpv
```

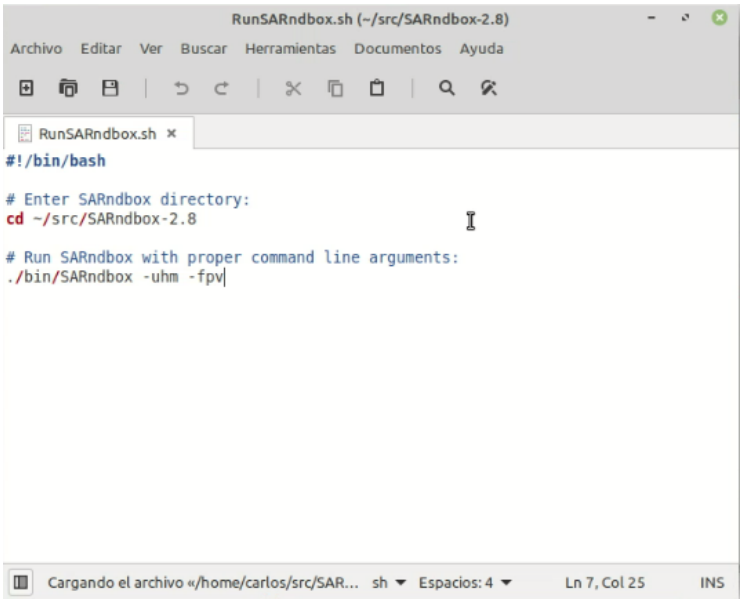

Figura 71: ventana del archivo script
Fuente: Autor

Una vez que se haya pegado estas líneas de comando se debe guardar y salir del editor de texto. En la misma ventana de la terminal sobre la cual estábamos trabajamos se lo debe ejecutar.

```
chmod a+x ~/src/SARndbox-2.8/RunSARndbox.sh
```

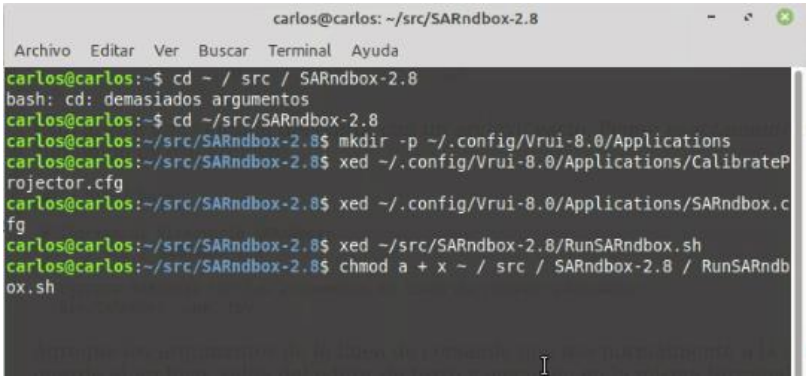

Figura 72: Comando para ejecutar el script
Fuente: Autor

Esto le dirá al sistema operativo que se puede ejecutar el script de Shell. Para crear un icono en el escritorio se debe ejecutar en una venta de terminal el siguiente comando.

xed ~/Desktop/RunSARndbox.desktop

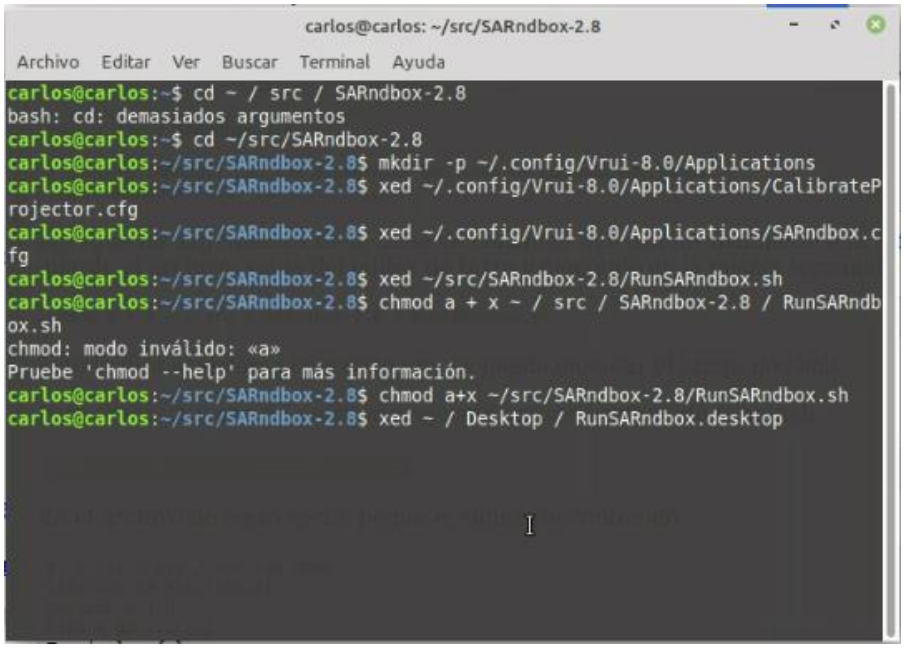

Figura 73: Comando para crear el icono en el escritorio
Fuente: Autor

De la misma forma, dentro del archivo de texto vacío, pegue el siguiente contenido.

#!/usr/bin/env xdg-open
[Desktop Entry]
Version=1.0
Type=Application
Terminal=false

Icon=mate-panel-launcher
Icon[en_US]=
Name[en_US]=
Exec=/home/<username>/src/SARndbox-2.8/RunSARndbox.sh
Comment[en_US]=
Name=Start the AR Sandbox
Comment=

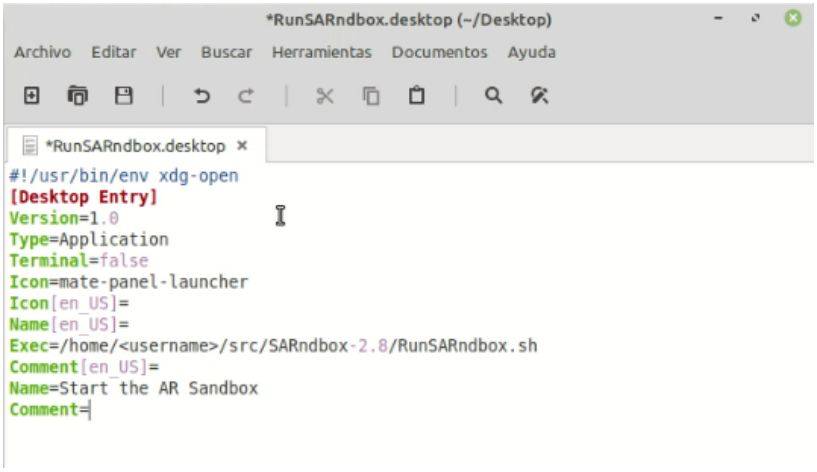

Figura 74: Archivo de texto para crear el icono
Fuente: Autor

Dentro de estas líneas de comando se debe remplazar el nombre de usuario, con su nombre real, para que se pueda ubicar el script de Shell creado en el paso anterior. Luego se guarda el archivo.

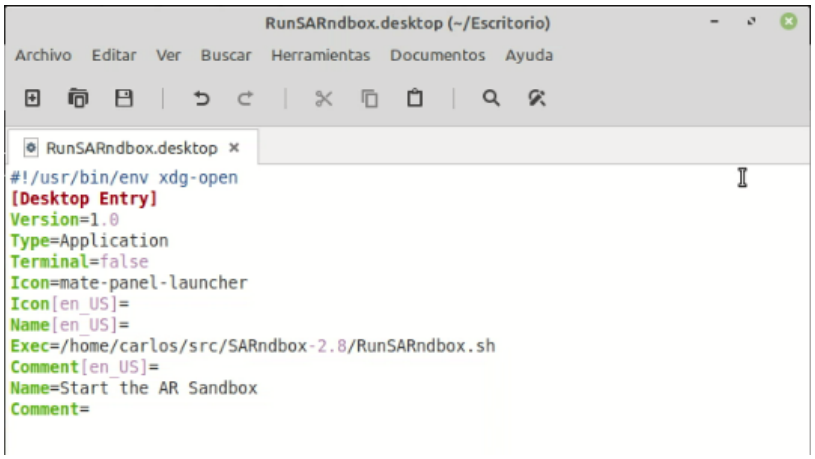

Figura 75: Modificación del nombre de usuario
Fuente: Autor

Para finalizar ejecute el siguiente comando para que el archivo se haga ejecutable.

chmod a+x ~/Desktop/RunSARndbox.desktop

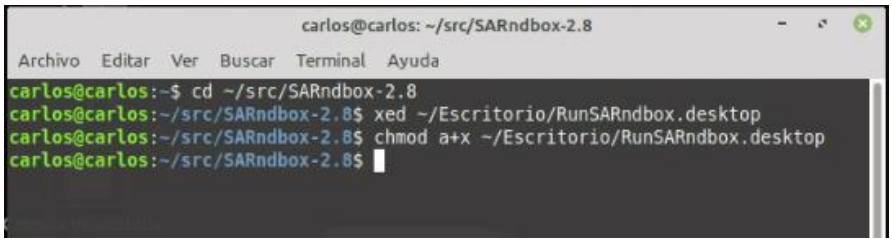

Figura 76: Comando para hacer el archivo ejecutable
Fuente: Autor

Ahora al hacer clic en el icono se abrirá inmediatamente el AR Sandbox.

Figura 77: Icono del AR Sandbox
Fuente: Autor

6.2. Diseño del simulador de mapa topográfico interactivo

6.2.1. Modelo base a seguir

El modelo base a seguir para la construcción es el que se muestra a continuación:

Figura 78: modelo base a seguir
Fuente:(Worthington Direct, 2021)

Este diseño fue desarrollado por los creadores originales del dispositivo considerando las distancias mínimas que se requieren para capturar y proyectar los relieves en la superficie de la arena. Tomando en cuenta el tiro del proyector y la cobertura de la cámara Kinect.

6.2.2. Prototipo

Mediante la utilización del software Solidworks se realizó el diseño de la caja de arena en base al modelo anteriormente señalado, se realizaron algunas modificaciones debido a los materiales del medio y de las necesidades que surgieron al momento de acoplar los dispositivos. A continuación, se presentará el prototipo visto desde varias perspectivas.

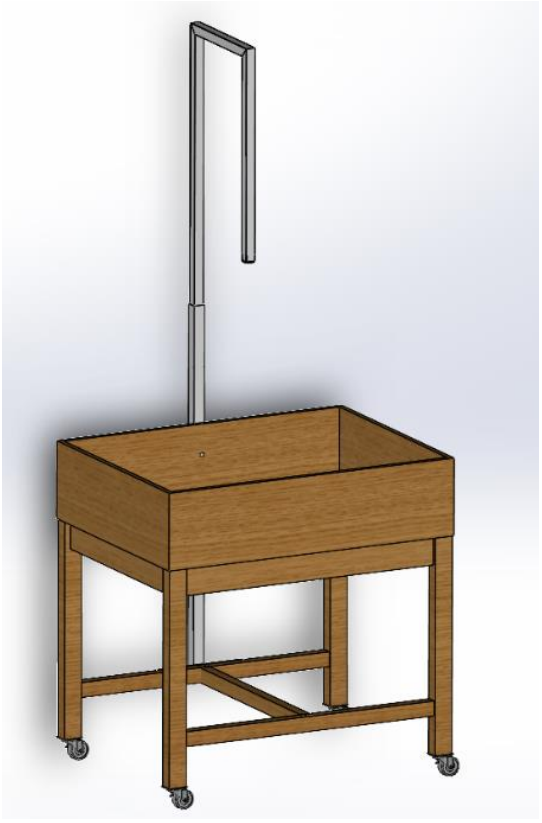

Figura 79: vista lateral izquierda de la caja de arena en Solidworks
Fuente: Autor

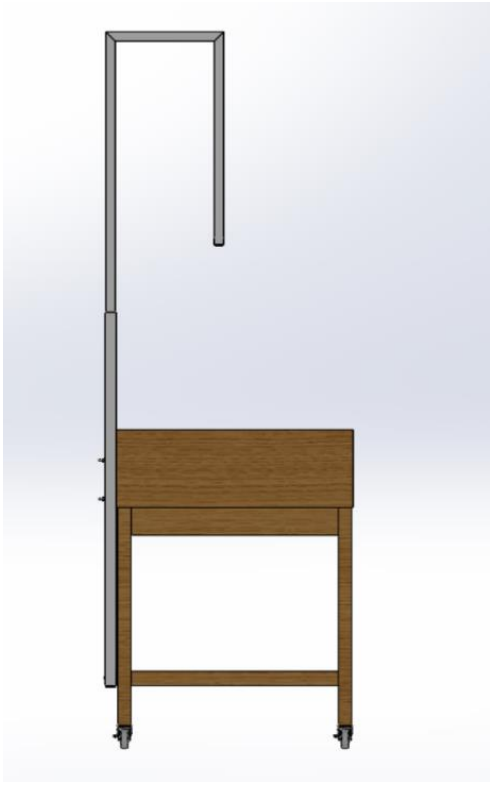

Figura 80: Vista lateral de la caja de arena en Solidworks
Fuente: Autor

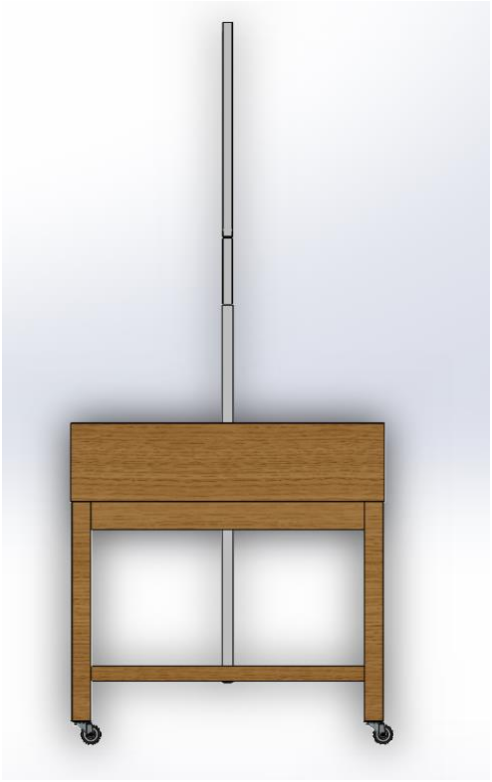

Figura 81: Vista frontal de la caja de arena en Solidworks
Fuente: Autor

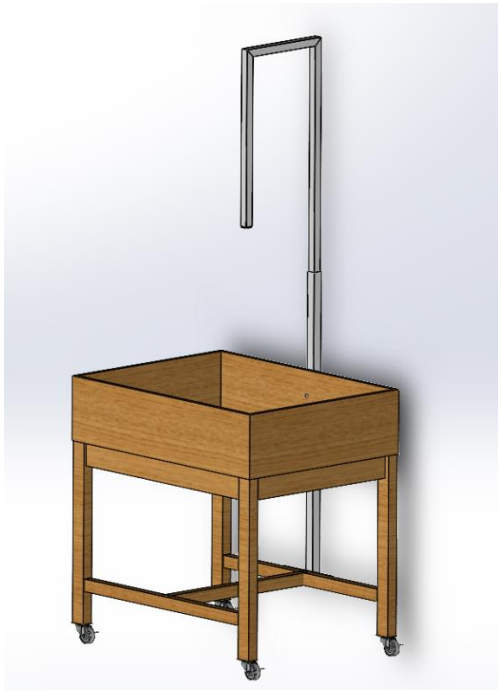

Figura 82: vista lateral derecha de la caja de arena en Solidworks
Fuente: Autor

Figura 83: vista lateral izquierda de la caja de arena en Solidworks
Fuente: Autor

6.3. Construcción de la estructura física de la caja de arena

Siguiendo el modelo base señalado anteriormente se realizó la elaboración de la estructura tomando en cuenta las medidas sugeridas. Cabe mencionar que la estructura se la trabajo en dos etapas, las cuales contemplan la elaboración de la caja de arena y la elaboración de la estructura de soporte de la cámara kinect y del proyector.

La estructura principal de la caja de arena se la trabajo con madera, ya que es un material resistente, fácil de moldear y acoplar. Esta fue lacada para una mayor resistencia a la humedad que se pueda producir por la arena.

Figura 84: soporte de caja de arena
Fuente: Autor

Como se mencionó en un principio se acoplaron las ruedas en la parte inferior de la caja de arena para darle mayor movilidad a la misma. Estas cuentan con seguros que le permiten permanecer estática en el lugar donde se la desee ubicar.

Figura 85: Acoplamiento de ruedas a la caja de arena
Fuente: Autor

La estructura terminada de la caja de arena se muestra en la figura 86, esta sería una parte de la estructura completa del simulador de mapa topográfico.

Figura 86: Estructura terminada de la caja de arena
Fuente: Autor

La parte secundaria se la elaboro con aluminio, la misma que está constituida por dos perfiles de aluminio para que su altura sea modificable. La altura puede ser ajustada mediante el desplazamiento de cada una de las partes y puede ser sujetada o fijada mediante los pernos que se ubican en la parte céntrica de los perfiles.

Figura 87: Estructura de Aluminio
Fuente: Autor

Los mismos cuentan con un conector de plástico en los vértices de los perfiles de aluminio para que soporte el peso de los equipos, en este caso del Kinect y del proyector.

Figura 88: Conector de plástico para perfiles de aluminio
Fuente: Autor

La estructura completa de la caja de arena se representa en la siguiente figura, esta muestra el acoplamiento de la estructura de madera con la estructura de aluminio.

Figura 89: Estructura completa de la caja de arena
Fuente: Autor

Se realizo un proceso de limpieza de la arena de la caja, esta se la hizo con la ayuda de un colador como se muestra en la figura 90, el cual permitió deshacernos de la arena gruesa e impurezas que no permitían una adecuada manipulación y modificación de la superficie de arena.

Figura 90: Limpieza de la arena
Fuente: Autor

Luego de haber acoplado los dispositivos en la estructura, se debe cuadrar la proyección, de tal manera que esta cubra la superficie en su totalidad, así como se muestra en la figura siguiente.

Figura 91: Encuadre de la proyección dentro de la superficie
Fuente: Autor

El acoplamiento del proyector se lo realizó mediante la adaptación de una estructura constituida por dos placas, como se muestra en la figura 92. Estas están sujetas con pernos en sitios estratégicos para lograr la estabilización.

Figura 92: Estructura para acoplamiento
Fuente: Autor

6.4. Pruebas de Funcionamiento

Los resultados del estudio e implementación de simulador de mapa topográfico interactivo para la facultad de energía de la UNL fueron determinados mediante la funcionalidad del prototipo.

El resultado de la funcionalidad de del prototipo de mapa topográfico interactivo se muestra en la siguiente gráfica, donde se observa la proyección del relieve que se moldea en la superficie de la arena, la cual ha sido modificada mediante la manipulación de la misma.

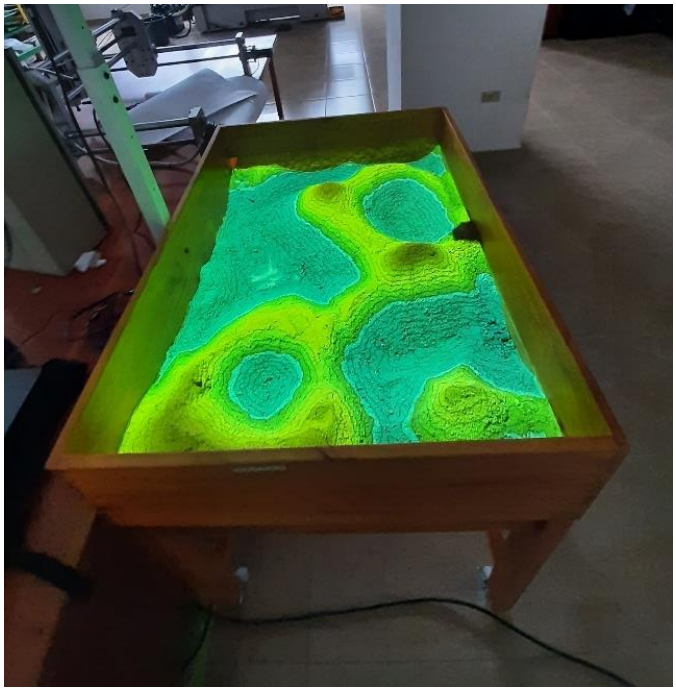

Figura 93: proyección Topográfica
Fuente: Autor

Como se observa este prototipo está constituido por una barra de aluminio, la misma que soporta el sensor Kinect y el proyector. Así mismo se puede apreciar la caja de arena, en la misma que se proyecta sobre la arena la imagen de los mapas generados por el software.

La superficie de la arena puede ser modificada para generar una diversidad de relieves, que van desde superficies muy bajas hasta elevaciones bastante considerables, esta superficie es modificada dependiendo de las necesidades de representación del usuario.

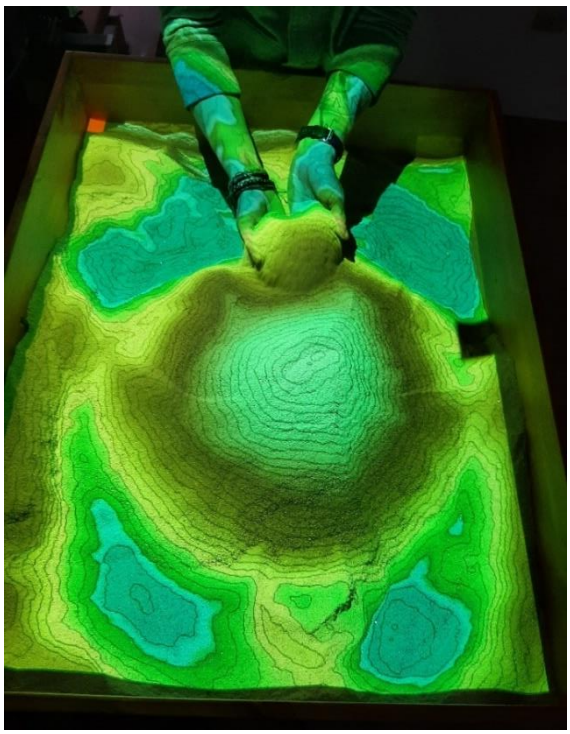

Figura 94: Modificación de la superficie para generar relieves
Fuente: Autor

Para poder replicar la topografía de una superficie montañosa se debe moldear la arena dándole una forma elevada. De tal manera que el sensor Kinect adquiera la información de la superficie, estos datos son procesados posteriormente por el programa y finalmente genera la proyección del relieve correspondiente sobre la arena.

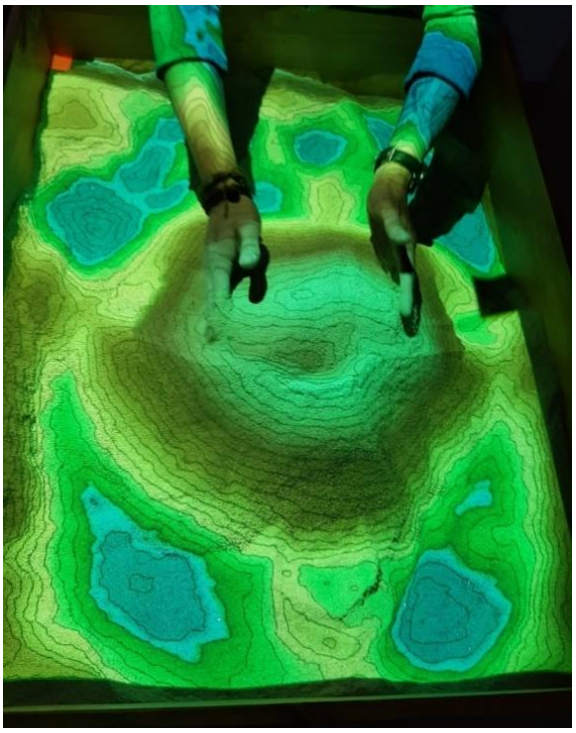

Figura 95: relieve de superficie montañosa
Fuente: Autor

Mediante la paleta de colores se representa las zonas más elevadas, es decir el color naranja si este fuera de gran elevación, por otro lado, el color azul representa las zonas más bajas, es decir el nivel del mar y a esto se agregan las líneas de color negro que representan las curvas de nivel.

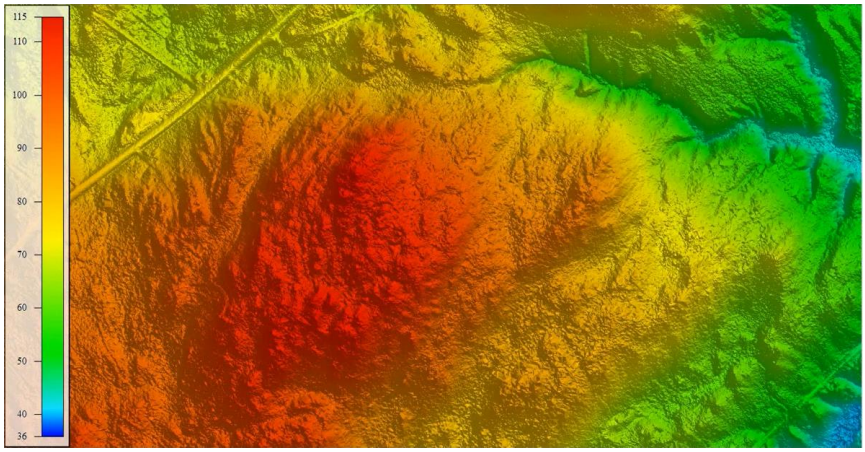

Figura 96: Representación de colores del relieve
Fuente: Autor

Es de suma importancia señalar que el tiempo de refresco o de actualización del relieve es casi inmediato, es decir al realizar algún cambio a la superficie este es procesado rápidamente y genera la proyección de manera casi instantánea.

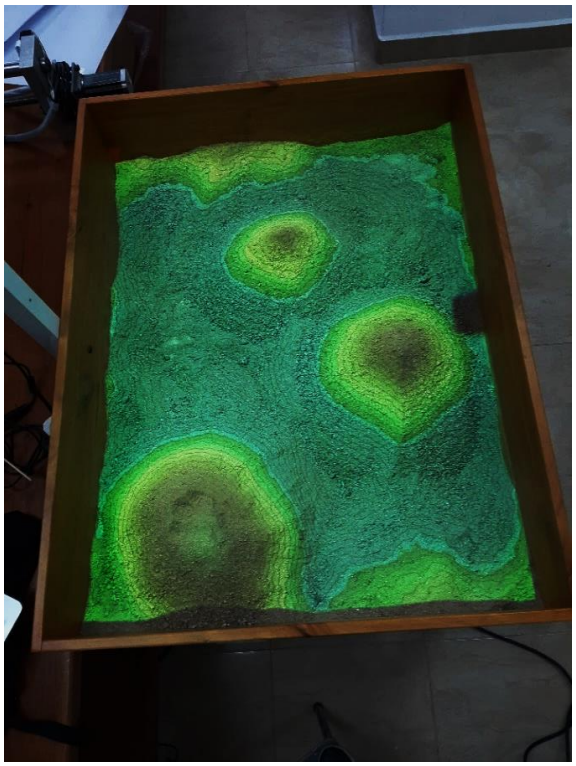

Figura 97: Representación de elevaciones
Fuente: Autor

Esta superficie puede ser moldeada por el usuario sin restricción alguna, de tal manera que se pueda replicar la topografía de un volcán, la arena fue moldeada de una forma creativa, para que contemple las características de un volcán, el sensor Kinect adquiere los datos de la distancia respecto a la caja de arena, por consiguiente, estos datos son procesados por el software, que a su vez cambia la proyección de la topografía en tiempo real.

6.5. Cálculo de porcentaje de diferencia

En este apartado se desarrolló el cálculo del porcentaje de diferencia (X) de las curvas de nivel proyectadas, para tratar de determinar la exactitud con la que están siendo proyectadas sobre la superficie, este cálculo se realizó mediante la siguiente ecuación:

$$X = \frac{|valor\ nuevo(B) - valor\ antiguo(A)|}{|valor\ antiguo(A)|} * 100\%$$

Tomando en cuenta dos puntos de análisis (A y B) de una determinada curva de nivel (figura 98) de la superficie de arena del prototipo, los mismo que tienen como referencia la base de madera de la caja de arena.

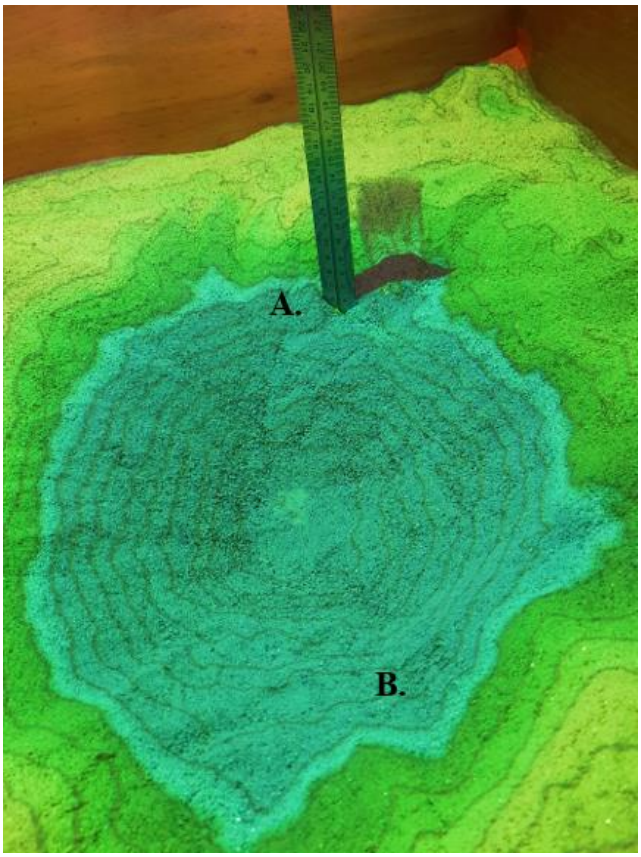

Figura 98: *Puntos de análisis*
Fuente: Autor

Como se aprecia en la figura 99 el valor del punto A es de 6 cm tomado desde la base de la caja de arena.

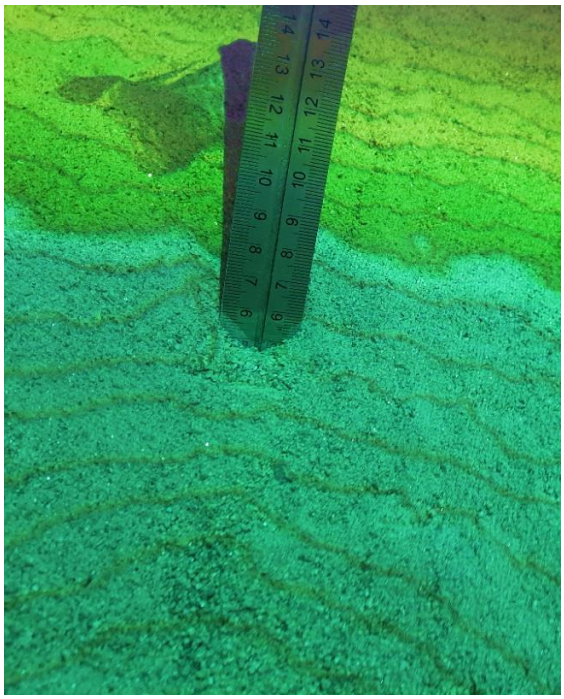

Figura 99: Valor del punto A
Fuente: Autor

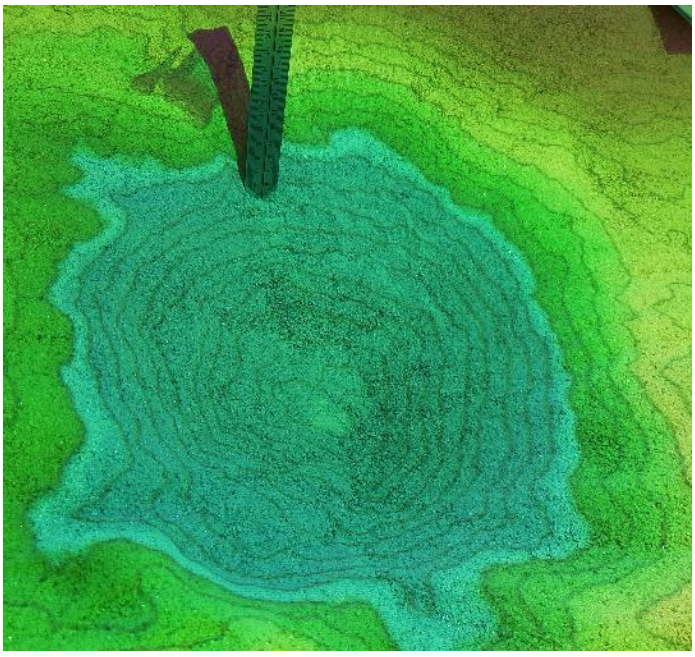

Figura 100: Ubicación del punto B
Fuente: Autor

De la misma forma en la figura 101 se observa el valor del punto B el cual es equivalente a 5,8 cm.

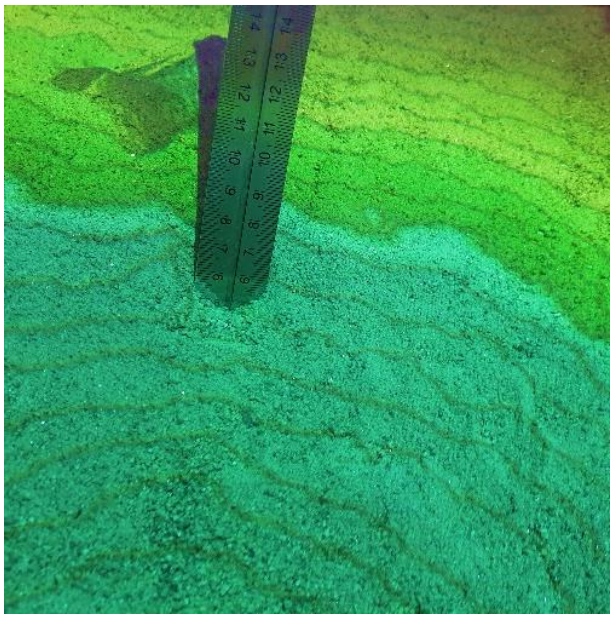

Figura 101:Valor del punto B
Fuente: Autor

En base a los datos de los puntos se obtuvo el siguiente resultado.

$$X = \frac{|5{,}8 - 6|}{|6|} * 100\%$$

$$X = 3{,}33\%$$

Como se puede observar el valor de porcentaje de diferencia de la curva de nivel es de 3,33 %, esto se debe a la estabilización del Kinect sobre el soporte de aluminio que puede variar según la manipulación de la estructura. Es necesario recalcar que este proceso se realizó empíricamente, es decir que está basado en la experiencia y observación de los hechos, en este caso la observación de las curvas de nivel proyectadas en la superficie.

Por otro lado, también se pueden dar casos donde algunas elevaciones con pendientes bastante pronunciadas(acantilados) le dificulten al Kinect la captura de las zonas que se encuentra por debajo de esta (figura 102).

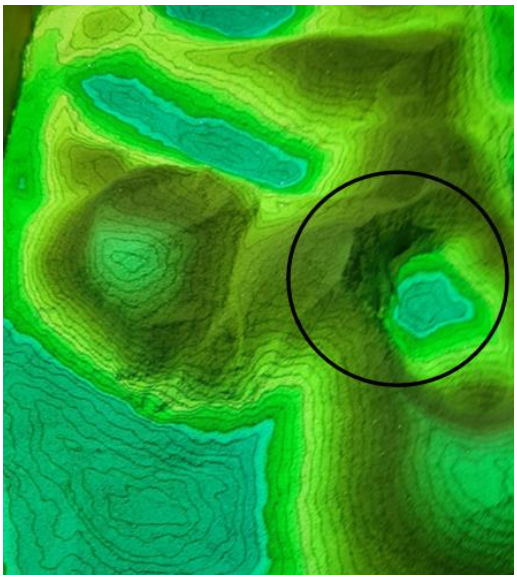

Figura 102: Pendiente con pendiente pronunciada (acantilado)
Fuente: Autor

7. Discusión

Con el objetivo de implementar un simulador de mapa topográfico que permitan a los docentes de la Facultad de Energía de la UNL mejorar los métodos educativos en el campo de las Ciencias de la Tierra, los resultados reflejan que este dispositivo viene a convertirse en un elemento potenciador de los espacios educativos, permitiendo crear nuevas formas de impartir los conocimientos a los estudiantes, apoyando a la formación académica. Esto quiere decir que los estudiantes van a poder desarrollar sus destrezas y habilidades de una forma más eficiente mediante el manejo de esta tecnología para poner en práctica los conocimientos conceptuales. Estos resultados son respaldados por (Bohórquez Melo, 2018) y (Sevilla, 2017) quienes consideran que con la realidad amentada podemos aprender de la observación fuera del aula,

siendo un concepto diferente de aprendizaje basado en el descubrimiento y es una herramienta muy útil dentro de la clase que ayuda a reforzar los conocimientos mediante la visualización de modelos 3D. A partir de múltiples opciones que ofrece el software dentro del campo de la geología, topografía, cartografía, etc. se puede desarrollar la creación de cuencas hidrográficas y regulación de caudal, áreas de captación, diques, como también la creación de superficies terrestres o la simulación a partir de maquetas hechas a escala. También se pueden emplear para distintos niveles educativos adaptando la complejidad de los contenidos planteados a las características de los estudiantes. Igualmente, al ser un software de código abierto, existe la posibilidad de agregar y programar fluidos personalizados por el usuario, como puede ser, fluido de agua, magma o petróleo para simular los impactos relacionados. Analizando estos resultados podemos corroborar que este dispositivo permite la implementación de diversas metodologías que permiten un enfoque integral de la docencia donde se integran conceptos, habilidades, destrezas, aptitudes y valores en el estudio de las Ciencias de la Tierra.

8. Conclusiones

- Mediante la recopilación y análisis de la información concerniente a los sistemas de simulación en 3D interactivos para topografía, se concluyó que el software SARnbox 2.8 es el más adecuado para este fin debido a su distribución libre y a su fácil acceso.

- Se puede concluir que la aplicación de la realidad aumentada en el ámbito educativo ha mostrado ser una técnica innovadora para estudiantes y maestros, debido a que mejora y refuerza los conocimientos mediante la visualización de modelos 3D y puede ser empleado en los diferentes niveles de enseñanza, mediante la adaptación de los contenidos planteados, dependiendo de la complejidad de los mismos.

- La implementación del simulador de mapa topográfico permite la creación de nuevas metodologías de enseñanza basadas en tecnologías innovadoras que le permiten al estudiante una mayor receptividad de los conocimientos impartidos teóricamente.

- Este recurso educativo disminuye considerablemente la desigualdad que existe para acceder a infraestructuras tecnologías en nuestra institución, mejorando en sobremanera el bienestar de los estudiantes y docentes.

- El desarrollo presente proyecto cumple con los objetivos propuestos, los cuales están centrados en el diseño y réplica de un simulador de mapa topográfico interactivo y mediante las comprobaciones realizadas del prototipo permitieron demostrar el correcto funcionamiento del simulador de mapa topográfico interactivo.

- Este prototipo contribuye a la potenciación de los espacios educativos de la Universidad Nacional de Loja, aportando al desarrollo de las destrezas de los estudiantes, las mismas que le proveerá un óptimo desenvolvimiento en el campo laboral.

- Con la construcción del prototipo se demostró que es posible la elaboración de esta herramienta de realidad aumentada con recursos que se pueden adquirir sin mayor dificultad.

9. Recomendaciones

- Tomar en cuenta la guía de uso y la guía de mantenimiento para el correcto manejo y cuidado del prototipo, que aumentara la vida útil del simulador.

- La captación de los puntos de unión se debe realizar con la proyección sobre la superficie, más no con el uso único de la pantalla.

- Si se requiere una mayor definición de la superficie se puede implementar una tarjeta gráfica de mayor gama.

- No verter ningún liquido sobre la superficie puesto que esto repercutiría desfavorablemente en la zona de la madera.

- Determinar las características del proyector antes de la construcción de estructura de metal, para determinar adecuadamente la altura a la cual el proyector debería ir para que su proyección cubra toda la superficie de la caja.

- Realizar el estudio que permita mejorar las propiedades físicas de la superficie moldeable. Así mismo se sugiere adaptar un soporte para el ordenador que permite una mayor movilidad del simulador.

10. Bibliografía

Álvarez, A. G. (2017). *Aplicaciones de la realidad aumentada en la promoción y la edificación: Vol. Universida*.

AR-sandbox.eu. (2020). *AR-sandbox.eu*. Licencia de KinectSandbox – Media. https://ar-sandbox.eu/product/kinectsandbox-license/

Bohórquez Melo, I. M. (2018). *Realidad aumentada y aplicaciones*.

Casanova, L. (2010). Levantamientos Topograficos. In *7* (p. 30).

Castillo Otegui, J. (2017). *LA REALIDAD VIRTUAL Y LA REALIDAD AUMENTADA*. 171.

Epson. (2021). *Proyector Inalámbrico PowerLite 2247U Full HD WUXGA 3LCD*. https://epson.com.ec/Para-el-trabajo/Proyectores/Salas-de-Reuniones/Proyector-Inalámbrico-PowerLite-2247U-Full-HD-WUXGA-3LCD/p/V11H881020

Fundación Telefónica. (2011). *Realidad Aumentada: una nueva lente para ver el mundo*.

Gonzáles Morcillos, Carlos; Fernández vallejo, D., Albusac Jiménez, J., & Castro Sánchez, J. J. (2012). *Realidad Aumentada: Un enfoque práctico con ARToolKit y Blender* (Bubok Publ). Escuela Superior de Informática.

Instituto Geografico Nacional del Gobierno de España. (2017). *Conceptos Cartográficos*. http://www.ign.es/web/resources/cartografiaEnsenanza/conceptosCarto/descargas/Conceptos_Cartograficos_def.pdf

Kim, H., & Choi, Y. (2019). *Desarrollo de una Interfaz de Usuario 3D basada en Sensor Kinect y Guante de Datos Bend-Sensing para Software de Control en la Industria Minera* [Universidad Nacional de Pukyong]. https://www.jksmer.or.kr/articles/xml/vJRV/

Magitech. (2020). *aptop Dell Inspiron 15 5567, Core i7-7500U 2.7GHz, RAM 8GB, HDD 1TB , Video 4GB ddr5 R7 M445, DVD, LED 15.6" HD, Windows 10*. https://www.magitech.pe/laptop-dell-inspiron-15-5567-core-i7-7500u-2-7ghz-ram-8gb-hdd-1tb-video-4gb-ddr5-r7-m445-dvd-led-15-6-hd-windows-10.html

Pascual Estapé, J. A. (2014). *Así es Kinect 2.0 para Windows en PC*. https://computerhoy.com/noticias/hardware/asi-es-kinect-20-windows-pc-10937

Pesce, M. (2020). *Realidad aumentada: pasado, presente y futuro*. https://www.interaction-design.org/literature/article/augmented-reality-the-past-the-present-and-the-future FG - 0

Reinoso, R. (2014). Introducción a la Realidad Aumentada. *Tecnotic*, 34.

Robles, J. G., & Escudero, E. R. (2017). *Mapas Topográficos*. https://formacion.uam.es/pluginfile.php/83821/mod_resource/content/1/PRACTICA6.pdf#:~:text=Un mapa topográfico es la,proyección planar de una esfera.

Ruiz Henry, A. C., Mesa Jiménez, F. Y., & Suarez Barón, M. J. (2018). *Realidad aumentada (RA): aplicaciones y desafíos para su uso en el aula de clase*. 148.

Sevilla, A. B. (2017). *Realidad Aumentada en Educación*. Sevilla, Alegría Blázqez.

UC Davis Department of Geology. (2022). *Augmented Reality Sandbox*. https://web.cs.ucdavis.edu/~okreylos/ResDev/SARndbox/

Vivas, A. A., Urrea, I. P. F., & Caballero, J. A. M. (2020). *Caja de arena, realidad aumentada en 3D*.

W3Schools. (2022). *Valor RGBA*. https://www.w3schools.com/css/css_colors_rgb.asp

Worthington Direct. (2021). *Worthington Direct*. https://www.worthingtondirect.com/storage-furniture/ars-4030-augmented-reality-sandbox.htm

11. Anexos

Anexo 1: Especificaciones Técnicas

Kinect

Figura 103:Kinect y adaptador
Fuente: Autor

Tabla 2: *Especificaciones técnicas kinect*
Fuente: (Pascual Estapé, 2014)

Parámetro	Especificaciones
Resolución	640x480 - 4:3
Ángulo de visión	57 grados horizontal, 43 grados vertical
Distancia media de uso	1.82 metros
Latencia	102 ms
Ajuste manual del motor	Sí
Detección simultanea de personas	4

Puntos del cuerpo simultáneos	20
Rango de inclinación vertical	± 27°
Velocidad de fotogramas	30 cuadros por segundo (FPS)
Formato de audio	Modulación de código de pulso mono (PCM) de 16 kHz y 24 bits

Proyector de video digital

Figura 104: proyector de video digital
Fuente: (Epson, 2021)

Tabla 3: *Especificaciones técnicas proyector de video digital*
Fuente: (Epson, 2021)

Parámetros	Especificaciones
Sistema de Proyección	Tecnología Epson 3LCD de 3 chips
Método de Proyección	Frontal / Posterior / Techo
Pantalla LCD	0.67-pulgadas (D10)
Método de visualización	Poly-silicon TFT active matrix
Número de Pixeles:	2,304,000 pixeles (1920 x 1200) x 3
Luminosidad en Color	4.200 Lúmenes
Luminosidad en Blanco	4.200 Lúmenes
Relación de Aspecto	16:10
Resolución Nativa	WUXGA (4:3)

Resolución Máxima Soportada	1920 x 1080 (1080p)
Re escalamiento	640 x 480, 800 x 600, 1280 x 1024, 1400 x 1050
Tipo de Lámpara	230 W UHE
Vida Útil de la Lámpara	5.500 horas (modo normal) 12.000 horas (modo Eco)
Relación de Proyección	1.39 (Zoom: Wide), 2.23 (Zoom: Tele)
Corrección de Trapecio	Vertical: -30° +30º Horizontal: -30° +30°
Dimensiones	309 x 90 x 282 mm (ancho x alto x profundidad)
Peso	3,2 Kg

Portátil

Figura 105: computadora portátil
Fuente: (Magitech, 2020)

Tabla 4: *Especificaciones técnicas computadora*
Fuente: (Magitech, 2020)

Parámetros	**Especificaciones**
Marca	DELL
Modelo	Inspiron 15 5000 Series 5567-8G

Procesador	Intel® Core™ i7-7500U 2.70GHz (4M Cache, hasta 3.50GHz) con Intel® Turbo Boost Technology 2.0-Séptima Generación.
Memoria RAM	8 GB DDR4 2400MHz ampliable a 32GB
Disco Duro	1TB (5400 RPM) Serial ATA
Pantalla	LED iluminada con TrueLife™ de 15.6" de alta definición (HD) (1366 x 768)
Tarjeta de video	AMD Radeon™ R5 M445 4GB DDR5
Conectividad	Ethernet (RJ-45) Wi-Fi 802.11b/g/n Dual-band 802.11AC
Multimedia	Cámara web HD 720P integrada Entrada para auriculares estéreo/micrófono
Puertos y Ranuras	HDMI USB 3.0 Lector de tarjetas de memoria Audio in/out
S.O.	Windows 10 Home en español
Batería	3 celdas - litio
Peso	2.32 kg

Anexo 2: Manual de operación y mantenimiento

Descripción general

El prototipo de simulador de mapa topográfico es un dispositivo de realidad aumentada que permite el modelamiento y determinación de las curvas de nivel de una superficie de trabajo a través del software AR Sandbox, mediante la manipulación de la arena se podrán crear relieves específicos, que posteriormente nos permitirá realizar el estudio de las características de la superficie creada.

Este es un dispositivo que no contempla gran riesgo de accidentes, sin embargo, es de suma importancia que la persona que opere este prototipo lea las indicaciones que se proporcionaran para que entienda el funcionamiento del simulador.

El inicio del dispositivo debe ser realizado por el encargado del simulador, el cual estará debidamente capacitado. Por ende, no permitir que personal no autorizado manipule los mismos.

No permita el consumo de ningún tipo de bebida o alimento cerca del área donde se encuentra el simulador.

No deje el prototipo encendido por un tiempo muy prolongado y sin supervisión.

Operación

Las instrucciones que se deben seguir para el uso y operación del simulador de mapa topográfico interactivo son las siguientes:

Encendido de AR Sandbox

1. Conectar el proyector, Kinect, y la computadora a la fuente de alimentación de energía eléctrica.
2. Conectar los dispositivos como el sensor Kinect y el proyector son conectados a los puertos de la tarjeta gráfica de la computadora.
3. Encienda el proyector haciendo clic en el botón de encendido.
4. Encienda la computadora presionando el botón encendido.
5. Esperar a que se inicie el sistema operativo Linux.

Inicio del programa Sandbox de realidad aumentada

1. Hace clic en el icono de “ARSandbox” que se encuentra en el escritorio.
2. Poner en pantalla completa presionando el botón F11 en el teclado.
3. Calibración rutinaria de Kinect:

a. Hacer dos montañas de arena distanciadas una de la otra en la caja de arena, las líneas de contorno deben formar círculos concéntricos que llevan a la cima de la montaña. Si los círculos no coinciden con las montañas, deberá ajustar el Kinect, al contrario, si la imagen coincide con la arena, no será necesario ningún ajuste.
b. Para realizar el ajuste de las líneas de contorno, deberá mover suavemente la parte superior del Kinect, exactamente donde se encuentran las cámaras, efectuar este movimiento en todos los ejes hasta que la imagen se alinee con la arena.

Apagado de AR Sandbox

1. Cerrar la ventana del ARSandbox haciendo clic en la esquina superior de la ventana.
2. Apagar la computadora, haciendo clic en el botón de apagado que se encuentra en el menú de inicio.
3. Apague el proyector presionando dos veces el botón de encendido.
4. Desconectar todos los dispositivos de la fuente de alimentación.

Mantenimiento

Las actividades de mantenimiento están destinadas a la conservación de los materiales constructivos y alargar el tiempo de vida útil del simulador mediante la limpieza y revisión continua del prototipo.

Mantenimiento de arena

1. Remover la arena con frecuencia para mantenerla seca.
2. De ser posible cubrir la superficie de la arena para mantenerla libre de cualquier imperfección que se pueda adherir a la misma.
3. Comprobar que la arena no este húmeda para evitar que la madera se deteriore.
4. Revisar si existen objetos extraños en la arena, de ser el caso retirarlos.

Mantenimiento de dispositivos

5. Revisar la distancia a la que se localice el sensor Kinect y que el mismo se encuentre centrado con referencia la superficie de la arena.

6. Examinar que el proyector se encuentre a la distancia correcta, de manera que su proyección cubra toda la superficie de la arena.
7. Comprobar que las líneas de contorno proyectadas formen círculos concéntricos en la superficie de las montañas moldeadas en la arena.

Anexo 3: Código en C++ del programa

El código que se muestra a continuación es adquirido en la página oficial de los desarrolladores científicos del centro keckCAVES de la universidad de California (https://web.cs.ucdavis.edu/~okreylos/ResDev/SARndbox/).

```
1. /**********************************************************************
2. Sandbox - Vrui application to drive an augmented reality sandbox.
3. Copyright (c) 2012-2019 Oliver Kreylos
4.
5. This file is part of the Augmented Reality Sandbox (SARndbox).
6.
7. The Augmented Reality Sandbox is free software; you can redistribute it
8. and/or modify it under the terms of the GNU General Public License as
```

```
9.  published by the Free Software Foundation; either version 2 of the
10. License, or (at your option) any later version.
11.
12. The Augmented Reality Sandbox is distributed in the hope that it will be
13. useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
14. MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
15. General Public License for more details.
16.
17. You should have received a copy of the GNU General Public License along
18. with the Augmented Reality Sandbox; if not, write to the Free Software
19. Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
20. ********************************************************************/
21.
22. #include "Sandbox.h"
23.
24. #include <ctype.h>
25. #include <string.h>
26. #include <stdlib.h>
27. #include <sys/types.h>
28. #include <sys/stat.h>
29. #include <unistd.h>
30. #include <fcntl.h>
31. #include <string>
32. #include <vector>
33. #include <stdexcept>
34. #include <iostream>
35. #include <Misc/SizedTypes.h>
36. #include <Misc/SelfDestructPointer.h>
37. #include <Misc/FixedArray.h>
38. #include <Misc/FunctionCalls.h>
39. #include <Misc/MessageLogger.h>
40. #include <Misc/FileNameExtensions.h>
41. #include <Misc/StandardValueCoders.h>
42. #include <Misc/ArrayValueCoders.h>
43. #include <Misc/ConfigurationFile.h>
44. #include <IO/File.h>
45. #include <IO/ValueSource.h>
46. #include <IO/OpenFile.h>
47. #include <Comm/OpenPipe.h>
48. #include <Math/Math.h>
49. #include <Math/Constants.h>
50. #include <Math/Interval.h>
51. #include <Math/MathValueCoders.h>
52. #include <Geometry/Point.h>
53. #include <Geometry/AffineCombiner.h>
54. #include <Geometry/HVector.h>
55. #include <Geometry/Plane.h>
56. #include <Geometry/LinearUnit.h>
57. #include <Geometry/GeometryValueCoders.h>
58. #include <Geometry/OutputOperators.h>
59. #include <GL/gl.h>
60. #include <GL/GLMaterialTemplates.h>
61. #include <GL/GLColorMap.h>
62. #include <GL/GLLightTracker.h>
63. #include <GL/Extensions/GLEXTFramebufferObject.h>
```

```
64. #include <GL/Extensions/GLARBTextureRectangle.h>
65. #include <GL/Extensions/GLARBTextureFloat.h>
66. #include <GL/Extensions/GLARBTextureRg.h>
67. #include <GL/Extensions/GLARBDepthTexture.h>
68. #include <GL/Extensions/GLARBShaderObjects.h>
69. #include <GL/Extensions/GLARBVertexShader.h>
70. #include <GL/Extensions/GLARBFragmentShader.h>
71. #include <GL/Extensions/GLARBMultitexture.h>
72. #include <GL/GLContextData.h>
73. #include <GL/GLGeometryWrappers.h>
74. #include <GL/GLTransformationWrappers.h>
75. #include <GLMotif/StyleSheet.h>
76. #include <GLMotif/WidgetManager.h>
77. #include <GLMotif/PopupMenu.h>
78. #include <GLMotif/Menu.h>
79. #include <GLMotif/PopupWindow.h>
80. #include <GLMotif/Margin.h>
81. #include <GLMotif/Label.h>
82. #include <GLMotif/TextField.h>
83. #include <Vrui/Vrui.h>
84. #include <Vrui/CoordinateManager.h>
85. #include <Vrui/Lightsource.h>
86. #include <Vrui/LightsourceManager.h>
87. #include <Vrui/Viewer.h>
88. #include <Vrui/ToolManager.h>
89. #include <Vrui/DisplayState.h>
90. #include <Kinect/FileFrameSource.h>
91. #include <Kinect/MultiplexedFrameSource.h>
92. #include <Kinect/DirectFrameSource.h>
93. #include <Kinect/OpenDirectFrameSource.h>
94.
95. #define SAVEDEPTH 0
96.
97. #if SAVEDEPTH
98. #include <Images/RGBImage.h>
99. #include <Images/WriteImageFile.h>
100. #endif
101.
102. #include "FrameFilter.h"
103. #include "DepthImageRenderer.h"
104. #include "ElevationColorMap.h"
105. #include "DEM.h"
106. #include "SurfaceRenderer.h"
107. #include "WaterTable2.h"
108. #include "HandExtractor.h"
109. #include "RemoteServer.h"
110. #include "WaterRenderer.h"
111. #include "GlobalWaterTool.h"
112. #include "LocalWaterTool.h"
113. #include "DEMTool.h"
114. #include "BathymetrySaverTool.h"
115.
116. #include "Config.h"
117.
118. /**********************************
```

```
119. Methods of class Sandbox::DataItem:
120. ***********************************/
121.
122. Sandbox::DataItem::DataItem(void)
123.         :waterTableTime(0.0),
124.          shadowFramebufferObject(0),shadowDepthTextureObject(0)
125.         {
126.         /* Check if all required extensions are supported: */
127.         bool supported=GLEXTFramebufferObject::isSupported();
128.         supported=supported&&GLARBTextureRectangle::isSupported();
129.         supported=supported&&GLARBTextureFloat::isSupported();
130.         supported=supported&&GLARBTextureRg::isSupported();
131.         supported=supported&&GLARBDepthTexture::isSupported();
132.         supported=supported&&GLARBShaderObjects::isSupported();
133.         supported=supported&&GLARBVertexShader::isSupported();
134.         supported=supported&&GLARBFragmentShader::isSupported();
135.         supported=supported&&GLARBMultitexture::isSupported();
136.         if(!supported)
137.                 Misc::throwStdErr("Sandbox: Not all required extensions are supported by local
    OpenGL");
138.
139.         /* Initialize all required extensions: */
140.         GLEXTFramebufferObject::initExtension();
141.         GLARBTextureRectangle::initExtension();
142.         GLARBTextureFloat::initExtension();
143.         GLARBTextureRg::initExtension();
144.         GLARBDepthTexture::initExtension();
145.         GLARBShaderObjects::initExtension();
146.         GLARBVertexShader::initExtension();
147.         GLARBFragmentShader::initExtension();
148.         GLARBMultitexture::initExtension();
149.         }
150.
151. Sandbox::DataItem::~DataItem(void)
152.         {
153.         /* Delete all shaders, buffers, and texture objects: */
154.         glDeleteFramebuffersEXT(1,&shadowFramebufferObject);
155.         glDeleteTextures(1,&shadowDepthTextureObject);
156.         }
157.
158. /****************************************
159. Methods of class Sandbox::RenderSettings:
160. ****************************************/
161.
162. Sandbox::RenderSettings::RenderSettings(void)
163.         :fixProjectorView(false),projectorTransform(PTransform::identity),projectorTransformV
    alid(false),
164.          hillshade(false),surfaceMaterial(GLMaterial::Color(1.0f,1.0f,1.0f)),
165.          useShadows(false),
166.          elevationColorMap(0),
167.          useContourLines(true),contourLineSpacing(0.75f),
168.          renderWaterSurface(false),waterOpacity(2.0f),
169.          surfaceRenderer(0),waterRenderer(0)
170.         {
171.         /* Load the default projector transformation: */
```

```
172.        loadProjectorTransform(CONFIG_DEFAULTPROJECTIONMATRIXFILENAME);
173.        }
174.
175. Sandbox::RenderSettings::RenderSettings(const Sandbox::RenderSettings& source)
176.        :fixProjectorView(source.fixProjectorView),projectorTransform(source.projectorTransfo
       rm),projectorTransformValid(source.projectorTransformValid),
177.         hillshade(source.hillshade),surfaceMaterial(source.surfaceMaterial),
178.         useShadows(source.useShadows),
179.         elevationColorMap(source.elevationColorMap!=0?new ElevationColorMap(*source.ele
       vationColorMap):0),
180.         useContourLines(source.useContourLines),contourLineSpacing(source.contourLineSpa
       cing),
181.         renderWaterSurface(source.renderWaterSurface),waterOpacity(source.waterOpacity),
182.         surfaceRenderer(0),waterRenderer(0)
183.        {
184.        }
185.
186. Sandbox::RenderSettings::~RenderSettings(void)
187.        {
188.        delete surfaceRenderer;
189.        delete waterRenderer;
190.        delete elevationColorMap;
191.        }
192.
193. void Sandbox::RenderSettings::loadProjectorTransform(const char* projectorTransformName
       )
194.        {
195.        std::string fullProjectorTransformName;
196.        try
197.              {
198.              /* Open the projector transformation file: */
199.              if(projectorTransformName[0]=='/')
200.                    {
201.                    /* Use the absolute file name directly: */
202.                    fullProjectorTransformName=projectorTransformName;
203.                    }
204.              else
205.                    {
206.                    /* Assemble a file name relative to the configuration file directory: */
207.                    fullProjectorTransformName=CONFIG_CONFIGDIR;
208.                    fullProjectorTransformName.push_back('/');
209.                    fullProjectorTransformName.append(projectorTransformName);
210.                    }
211.              IO::FilePtr projectorTransformFile=IO::openFile(fullProjectorTransformName.c_st
       r(),IO::File::ReadOnly);
212.              projectorTransformFile->setEndianness(Misc::LittleEndian);
213.
214.              /* Read the projector transformation matrix from the binary file: */
215.              Misc::Float64 pt[16];
216.              projectorTransformFile->read(pt,16);
217.              projectorTransform=PTransform::fromRowMajor(pt);
218.
219.              projectorTransformValid=true;
220.              }
221.        catch(const std::runtime_error& err)
```

```
222.                {
223.                /* Print an error message and disable calibrated projections: */
224.                std::cerr<<"Unable to load projector transformation from file
    "<<fullProjectorTransformName<<" due to exception "<<err.what()<<std::endl;
225.                projectorTransformValid=false;
226.                }
227.        }
228.
229. void Sandbox::RenderSettings::loadHeightMap(const char* heightMapName)
230.        {
231.        try
232.                {
233.                /* Load the elevation color map of the given name: */
234.                ElevationColorMap* newElevationColorMap=new ElevationColorMap(heightMap
    Name);
235.
236.                /* Delete the previous elevation color map and assign the new one: */
237.                delete elevationColorMap;
238.                elevationColorMap=newElevationColorMap;
239.                }
240.        catch(const std::runtime_error& err)
241.                {
242.                std::cerr<<"Ignoring height map due to exception "<<err.what()<<std::endl;
243.                }
244.        }
245.
246. /*************************
247. Methods of class Sandbox:
248. *************************/
249.
250. void Sandbox::rawDepthFrameDispatcher(const Kinect::FrameBuffer& frameBuffer)
251.        {
252.        /* Pass the received frame to the frame filter and the hand extractor: */
253.        if(frameFilter!=0&&!pauseUpdates)
254.                frameFilter->receiveRawFrame(frameBuffer);
255.        if(handExtractor!=0)
256.                handExtractor->receiveRawFrame(frameBuffer);
257.        }
258.
259. void Sandbox::receiveFilteredFrame(const Kinect::FrameBuffer& frameBuffer)
260.        {
261.        /* Put the new frame into the frame input buffer: */
262.        filteredFrames.postNewValue(frameBuffer);
263.
264.        /* Wake up the foreground thread: */
265.        Vrui::requestUpdate();
266.        }
267.
268. void Sandbox::toggleDEM(DEM* dem)
269.        {
270.        /* Check if this is the active DEM: */
271.        if(activeDem==dem)
272.                {
273.                /* Deactivate the currently active DEM: */
274.                activeDem=0;
```

```
275.                }
276.        else
277.                {
278.                /* Activate this DEM: */
279.                activeDem=dem;
280.                }
281.
282.        /* Enable DEM matching in all surface renderers that use a fixed projector matrix, i.e., in
    all physical sandboxes: */
283.        for(std::vector<RenderSettings>::iterator rsIt=renderSettings.begin();rsIt!=renderSettings
    .end();++rsIt)
284.                if(rsIt->fixProjectorView)
285.                        rsIt->surfaceRenderer->setDem(activeDem);
286.        }
287.
288. void Sandbox::addWater(GLContextData& contextData) const
289.        {
290.        /* Check if the most recent rain object list is not empty: */
291.        if(handExtractor!=0&&!handExtractor->getLockedExtractedHands().empty())
292.                {
293.                /* Render all rain objects into the water table: */
294.                glPushAttrib(GL_ENABLE_BIT);
295.                glDisable(GL_CULL_FACE);
296.
297.                /* Create a local coordinate frame to render rain disks: */
298.                Vector z=waterTable->getBaseTransform().inverseTransform(Vector(0,0,1));
299.                Vector x=Geometry::normal(z);
300.                Vector y=Geometry::cross(z,x);
301.                x.normalize();
302.                y.normalize();
303.
304.                glVertexAttrib1fARB(1,rainStrength/waterSpeed);
305.                for(HandExtractor::HandList::const_iterator hIt=handExtractor-
    >getLockedExtractedHands().begin();hIt!=handExtractor-
    >getLockedExtractedHands().end();++hIt)
306.                        {
307.                        /* Render a rain disk approximating the hand: */
308.                        glBegin(GL_POLYGON);
309.                        for(int i=0;i<32;++i)
310.                                {
311.                                Scalar
    angle=Scalar(2)*Math::Constants<Scalar>::pi*Scalar(i)/Scalar(32);
312.                                glVertex(hIt->center+x*(Math::cos(angle)*hIt-
    >radius*0.75)+y*(Math::sin(angle)*hIt->radius*0.75));
313.                                }
314.                        glEnd();
315.                        }
316.
317.                glPopAttrib();
318.                }
319.        }
320.
321. void Sandbox::pauseUpdatesCallback(GLMotif::ToggleButton::ValueChangedCallbackData*
     cbData)
322.        {
```

```
323.        pauseUpdates=cbData->set;
324.        }
325.
326. void Sandbox::showWaterControlDialogCallback(Misc::CallbackData* cbData)
327.        {
328.        Vrui::popupPrimaryWidget(waterControlDialog);
329.        }
330.
331. void Sandbox::waterSpeedSliderCallback(GLMotif::TextFieldSlider::ValueChangedCallback
     Data* cbData)
332.        {
333.        waterSpeed=cbData->value;
334.        }
335.
336. void Sandbox::waterMaxStepsSliderCallback(GLMotif::TextFieldSlider::ValueChangedCallb
     ackData* cbData)
337.        {
338.        waterMaxSteps=int(Math::floor(cbData->value+0.5));
339.        }
340.
341. void Sandbox::waterAttenuationSliderCallback(GLMotif::TextFieldSlider::ValueChangedCal
     lbackData* cbData)
342.        {
343.        waterTable->setAttenuation(GLfloat(1.0-cbData->value));
344.        }
345.
346. GLMotif::PopupMenu* Sandbox::createMainMenu(void)
347.        {
348.        /* Create a popup shell to hold the main menu: */
349.        GLMotif::PopupMenu* mainMenuPopup=new GLMotif::PopupMenu("MainMenuPopu
     p",Vrui::getWidgetManager());
350.        mainMenuPopup->setTitle("AR Sandbox");
351.
352.        /* Create the main menu itself: */
353.        GLMotif::Menu* mainMenu=new GLMotif::Menu("MainMenu",mainMenuPopup,false)
     ;
354.
355.        /* Create a button to pause topography updates: */
356.        pauseUpdatesToggle=new GLMotif::ToggleButton("PauseUpdatesToggle",mainMenu,"
     Pause Topography");
357.        pauseUpdatesToggle->setToggle(false);
358.        pauseUpdatesToggle-
     >getValueChangedCallbacks().add(this,&Sandbox::pauseUpdatesCallback);
359.
360.        if(waterTable!=0)
361.                {
362.                /* Create a button to show the water control dialog: */
363.                GLMotif::Button* showWaterControlDialogButton=new GLMotif::Button("Show
     WaterControlDialogButton",mainMenu,"Show Water Simulation Control");
364.                showWaterControlDialogButton-
     >getSelectCallbacks().add(this,&Sandbox::showWaterControlDialogCallback);
365.                }
366.
367.        /* Finish building the main menu: */
368.        mainMenu->manageChild();
```

```
369.
370.        return mainMenuPopup;
371.        }
372.
373. GLMotif::PopupWindow* Sandbox::createWaterControlDialog(void)
374.        {
375.        const GLMotif::StyleSheet& ss=*Vrui::getUiStyleSheet();
376.
377.        /* Create a popup window shell: */
378.        GLMotif::PopupWindow* waterControlDialogPopup=new GLMotif::PopupWindow("W
     aterControlDialogPopup",Vrui::getWidgetManager(),"Water Simulation Control");
379.        waterControlDialogPopup->setCloseButton(true);
380.        waterControlDialogPopup->setResizableFlags(true,false);
381.        waterControlDialogPopup->popDownOnClose();
382.
383.        GLMotif::RowColumn* waterControlDialog=new GLMotif::RowColumn("WaterContro
     lDialog",waterControlDialogPopup,false);
384.        waterControlDialog->setOrientation(GLMotif::RowColumn::VERTICAL);
385.        waterControlDialog->setPacking(GLMotif::RowColumn::PACK_TIGHT);
386.        waterControlDialog->setNumMinorWidgets(2);
387.
388.        new GLMotif::Label("WaterSpeedLabel",waterControlDialog,"Speed");
389.
390.        waterSpeedSlider=new GLMotif::TextFieldSlider("WaterSpeedSlider",waterControlDial
     og,8,ss.fontHeight*10.0f);
391.        waterSpeedSlider->getTextField()->setFieldWidth(7);
392.        waterSpeedSlider->getTextField()->setPrecision(4);
393.        waterSpeedSlider->getTextField()->setFloatFormat(GLMotif::TextField::SMART);
394.        waterSpeedSlider->setSliderMapping(GLMotif::TextFieldSlider::EXP10);
395.        waterSpeedSlider->setValueRange(0.001,10.0,0.05);
396.        waterSpeedSlider->getSlider()->addNotch(0.0f);
397.        waterSpeedSlider->setValue(waterSpeed);
398.        waterSpeedSlider-
     >getValueChangedCallbacks().add(this,&Sandbox::waterSpeedSliderCallback);
399.
400.        new GLMotif::Label("WaterMaxStepsLabel",waterControlDialog,"Max Steps");
401.
402.        waterMaxStepsSlider=new GLMotif::TextFieldSlider("WaterMaxStepsSlider",waterCon
     trolDialog,8,ss.fontHeight*10.0f);
403.        waterMaxStepsSlider->getTextField()->setFieldWidth(7);
404.        waterMaxStepsSlider->getTextField()->setPrecision(0);
405.        waterMaxStepsSlider->getTextField()->setFloatFormat(GLMotif::TextField::FIXED);
406.        waterMaxStepsSlider->setSliderMapping(GLMotif::TextFieldSlider::LINEAR);
407.        waterMaxStepsSlider->setValueType(GLMotif::TextFieldSlider::UINT);
408.        waterMaxStepsSlider->setValueRange(0,200,1);
409.        waterMaxStepsSlider->setValue(waterMaxSteps);
410.        waterMaxStepsSlider-
     >getValueChangedCallbacks().add(this,&Sandbox::waterMaxStepsSliderCallback);
411.
412.        new GLMotif::Label("FrameRateLabel",waterControlDialog,"Frame Rate");
413.
414.        GLMotif::Margin* frameRateMargin=new GLMotif::Margin("FrameRateMargin",water
     ControlDialog,false);
415.        frameRateMargin->setAlignment(GLMotif::Alignment::LEFT);
416.
```

```
417.        frameRateTextField=new GLMotif::TextField("FrameRateTextField",frameRateMargin,
    8);
418.        frameRateTextField->setFieldWidth(7);
419.        frameRateTextField->setPrecision(2);
420.        frameRateTextField->setFloatFormat(GLMotif::TextField::FIXED);
421.        frameRateTextField->setValue(0.0);
422.
423.        frameRateMargin->manageChild();
424.
425.        new GLMotif::Label("WaterAttenuationLabel",waterControlDialog,"Attenuation");
426.
427.        waterAttenuationSlider=new GLMotif::TextFieldSlider("WaterAttenuationSlider",water
    ControlDialog,8,ss.fontHeight*10.0f);
428.        waterAttenuationSlider->getTextField()->setFieldWidth(7);
429.        waterAttenuationSlider->getTextField()->setPrecision(5);
430.        waterAttenuationSlider->getTextField()-
    >setFloatFormat(GLMotif::TextField::SMART);
431.        waterAttenuationSlider->setSliderMapping(GLMotif::TextFieldSlider::EXP10);
432.        waterAttenuationSlider->setValueRange(0.001,1.0,0.01);
433.        waterAttenuationSlider->getSlider()->addNotch(Math::log10(1.0-double(waterTable-
    >getAttenuation())));
434.        waterAttenuationSlider->setValue(1.0-double(waterTable->getAttenuation()));
435.        waterAttenuationSlider-
    >getValueChangedCallbacks().add(this,&Sandbox::waterAttenuationSliderCallback);
436.
437.        waterControlDialog->manageChild();
438.
439.        return waterControlDialogPopup;
440.        }
441.
442. namespace {
443.
444. /*****************
445. Helper functions:
446. *****************/
447.
448. void printUsage(void)
449.        {
450.        std::cout<<"Usage: SARndbox [option 1] ... [option n]"<<std::endl;
451.        std::cout<<"  Options:"<<std::endl;
452.        std::cout<<"  -h"<<std::endl;
453.        std::cout<<"     Prints this help message"<<std::endl;
454.        std::cout<<"  -remote [<listening port ID>]"<<std::endl;
455.        std::cout<<"     Creates a data streaming server listening on TCP port <listening port
    ID>"<<std::endl;
456.        std::cout<<"     Default listening port ID: 26000"<<std::endl;
457.        std::cout<<"  -c <camera index>"<<std::endl;
458.        std::cout<<"     Selects the local 3D camera of the given index (0: first
    camera"<<std::endl;
459.        std::cout<<"     on USB bus)"<<std::endl;
460.        std::cout<<"     Default: 0"<<std::endl;
461.        std::cout<<"  -f <frame file name prefix>"<<std::endl;
462.        std::cout<<"     Reads a pre-recorded 3D video stream from a pair of
    color/depth"<<std::endl;
463.        std::cout<<"     files of the given file name prefix"<<std::endl;
```

```
464.        std::cout<<"  -s <scale factor>"<<std::endl;
465.        std::cout<<"     Scale factor from real sandbox to simulated terrain"<<std::endl;
466.        std::cout<<"     Default: 100.0 (1:100 scale, 1cm in sandbox is 1m in terrain"<<std::endl;
467.        std::cout<<"  -slf <sandbox layout file name>"<<std::endl;
468.        std::cout<<"     Loads the sandbox layout file of the given name"<<std::endl;
469.        std::cout<<"     Default:
    "<<CONFIG_CONFIGDIR<<'/'<<CONFIG_DEFAULTBOXLAYOUTFILENAME<<std::en
    dl;
470.        std::cout<<"  -er <min elevation> <max elevation>"<<std::endl;
471.        std::cout<<"     Sets the range of valid sand surface elevations relative to the"<<std::endl;
472.        std::cout<<"     ground plane in cm"<<std::endl;
473.        std::cout<<"     Default: Range of elevation color map"<<std::endl;
474.        std::cout<<"  -hmp <x> <y> <z> <offset>"<<std::endl;
475.        std::cout<<"     Sets an explicit base plane equation to use for height color
    mapping"<<std::endl;
476.        std::cout<<"  -nas <num averaging slots>"<<std::endl;
477.        std::cout<<"     Sets the number of averaging slots in the frame filter; latency
    is"<<std::endl;
478.        std::cout<<"     <num averaging slots> * 1/30 s"<<std::endl;
479.        std::cout<<"     Default: 30"<<std::endl;
480.        std::cout<<"  -sp <min num samples> <max variance>"<<std::endl;
481.        std::cout<<"     Sets the frame filter parameters minimum number of valid
    samples"<<std::endl;
482.        std::cout<<"     and maximum sample variance before convergence"<<std::endl;
483.        std::cout<<"     Default: 10 2"<<std::endl;
484.        std::cout<<"  -he <hysteresis envelope>"<<std::endl;
485.        std::cout<<"     Sets the size of the hysteresis envelope used for jitter
    removal"<<std::endl;
486.        std::cout<<"     Default: 0.1"<<std::endl;
487.        std::cout<<"  -wts <water grid width> <water grid height>"<<std::endl;
488.        std::cout<<"     Sets the width and height of the water flow simulation grid"<<std::endl;
489.        std::cout<<"     Default: 640 480"<<std::endl;
490.        std::cout<<"  -ws <water speed> <water max steps>"<<std::endl;
491.        std::cout<<"     Sets the relative speed of the water simulation and the
    maximum"<<std::endl;
492.        std::cout<<"     number of simulation steps per frame"<<std::endl;
493.        std::cout<<"     Default: 1.0 30"<<std::endl;
494.        std::cout<<"  -rer <min rain elevation> <max rain elevation>"<<std::endl;
495.        std::cout<<"     Sets the elevation range of the rain cloud level relative to the"<<std::endl;
496.        std::cout<<"     ground plane in cm"<<std::endl;
497.        std::cout<<"     Default: Above range of elevation color map"<<std::endl;
498.        std::cout<<"  -rs <rain strength>"<<std::endl;
499.        std::cout<<"     Sets the strength of global or local rainfall in cm/s"<<std::endl;
500.        std::cout<<"     Default: 0.25"<<std::endl;
501.        std::cout<<"  -evr <evaporation rate>"<<std::endl;
502.        std::cout<<"     Water evaporation rate in cm/s"<<std::endl;
503.        std::cout<<"     Default: 0.0"<<std::endl;
504.        std::cout<<"  -dds <DEM distance scale>"<<std::endl;
505.        std::cout<<"     DEM matching distance scale factor in cm"<<std::endl;
506.        std::cout<<"     Default: 1.0"<<std::endl;
507.        std::cout<<"  -wi <window index>"<<std::endl;
508.        std::cout<<"     Sets the zero-based index of the display window to which
    the"<<std::endl;
509.        std::cout<<"     following rendering settings are applied"<<std::endl;
510.        std::cout<<"     Default: 0"<<std::endl;
```

```
511.        std::cout<<"  -fpv [projector transform file name]"<<std::endl;
512.        std::cout<<"     Fixes the navigation transformation so that Kinect camera
    and"<<std::endl;
513.        std::cout<<"     projector are aligned, as defined by the projector transform
    file"<<std::endl;
514.        std::cout<<"     of the given name"<<std::endl;
515.        std::cout<<"     Default projector transform file name:
    "<<CONFIG_CONFIGDIR<<'/'<<CONFIG_DEFAULTPROJECTIONMATRIXFILENAME
    <<std::endl;
516.        std::cout<<"  -nhs"<<std::endl;
517.        std::cout<<"     Disables hill shading"<<std::endl;
518.        std::cout<<"  -uhs"<<std::endl;
519.        std::cout<<"     Enables hill shading"<<std::endl;
520.        std::cout<<"  -ns"<<std::endl;
521.        std::cout<<"     Disables shadows"<<std::endl;
522.        std::cout<<"  -us"<<std::endl;
523.        std::cout<<"     Enables shadows"<<std::endl;
524.        std::cout<<"  -nhm"<<std::endl;
525.        std::cout<<"     Disables elevation color mapping"<<std::endl;
526.        std::cout<<"  -uhm [elevation color map file name]"<<std::endl;
527.        std::cout<<"     Enables elevation color mapping and loads the elevation color map
    from"<<std::endl;
528.        std::cout<<"     the file of the given name"<<std::endl;
529.        std::cout<<"     Default elevation color  map file name:
    "<<CONFIG_CONFIGDIR<<'/'<<CONFIG_DEFAULTHEIGHTCOLORMAPFILENAME<
    <std::endl;
530.        std::cout<<"  -ncl"<<std::endl;
531.        std::cout<<"     Disables topographic contour lines"<<std::endl;
532.        std::cout<<"  -ucl [contour line spacing]"<<std::endl;
533.        std::cout<<"     Enables topographic contour lines and sets the elevation distance
    between"<<std::endl;
534.        std::cout<<"     adjacent contour lines to the given value in cm"<<std::endl;
535.        std::cout<<"     Default contour line spacing: 0.75"<<std::endl;
536.        std::cout<<"  -rws"<<std::endl;
537.        std::cout<<"     Renders water surface as geometric surface"<<std::endl;
538.        std::cout<<"  -rwt"<<std::endl;
539.        std::cout<<"     Renders water surface as texture"<<std::endl;
540.        std::cout<<"  -wo <water opacity>"<<std::endl;
541.        std::cout<<"     Sets the water depth at which water appears opaque in cm"<<std::endl;
542.        std::cout<<"     Default: 2.0"<<std::endl;
543.        std::cout<<"  -cp <control pipe name>"<<std::endl;
544.        std::cout<<"     Sets the name of a named POSIX pipe from which to read control
    commands"<<std::endl;
545.        }
546.
547. }
548.
549. Sandbox::Sandbox(int& argc,char**& argv)
550.        :Vrui::Application(argc,argv),
551.         remoteServer(0),
552.         camera(0),pixelDepthCorrection(0),
553.         frameFilter(0),pauseUpdates(false),
554.         depthImageRenderer(0),
555.         waterTable(0),
556.         handExtractor(0),addWaterFunction(0),addWaterFunctionRegistered(false),
```

```
557.         sun(0),
558.         activeDem(0),
559.         mainMenu(0),pauseUpdatesToggle(0),waterControlDialog(0),
560.         waterSpeedSlider(0),waterMaxStepsSlider(0),frameRateTextField(0),waterAttenuationS
     lider(0),
561.         controlPipeFd(-1)
562.         {
563.         /* Read the sandbox's default configuration parameters: */
564.         std::string sandboxConfigFileName=CONFIG_CONFIGDIR;
565.         sandboxConfigFileName.push_back('/');
566.         sandboxConfigFileName.append(CONFIG_DEFAULTCONFIGFILENAME);
567.         Misc::ConfigurationFile sandboxConfigFile(sandboxConfigFileName.c_str());
568.         Misc::ConfigurationFileSection cfg=sandboxConfigFile.getSection("/SARndbox");
569.         unsigned int cameraIndex=cfg.retrieveValue<int>("./cameraIndex",0);
570.         std::string cameraConfiguration=cfg.retrieveString("./cameraConfiguration","Camera");
571.         double scale=cfg.retrieveValue<double>("./scaleFactor",100.0);
572.         std::string sandboxLayoutFileName=CONFIG_CONFIGDIR;
573.         sandboxLayoutFileName.push_back('/');
574.         sandboxLayoutFileName.append(CONFIG_DEFAULTBOXLAYOUTFILENAME);
575.         sandboxLayoutFileName=cfg.retrieveString("./sandboxLayoutFileName",sandboxLayou
     tFileName);
576.         Math::Interval<double> elevationRange=cfg.retrieveValue<Math::Interval<double> >(".
     /elevationRange",Math::Interval<double>(-1000.0,1000.0));
577.         bool haveHeightMapPlane=cfg.hasTag("./heightMapPlane");
578.         Plane heightMapPlane;
579.         if(haveHeightMapPlane)
580.               heightMapPlane=cfg.retrieveValue<Plane>("./heightMapPlane");
581.         unsigned int numAveragingSlots=cfg.retrieveValue<unsigned int>("./numAveragingSlot
     s",30);
582.         unsigned int minNumSamples=cfg.retrieveValue<unsigned int>("./minNumSamples",10
     );
583.         unsigned int maxVariance=cfg.retrieveValue<unsigned int>("./maxVariance",2);
584.         float hysteresis=cfg.retrieveValue<float>("./hysteresis",0.1f);
585.         Misc::FixedArray<unsigned int,2> wtSize;
586.         wtSize[0]=640;
587.         wtSize[1]=480;
588.         wtSize=cfg.retrieveValue<Misc::FixedArray<unsigned int,2> >("./waterTableSize",wtSi
     ze);
589.         waterSpeed=cfg.retrieveValue<double>("./waterSpeed",1.0);
590.         waterMaxSteps=cfg.retrieveValue<unsigned int>("./waterMaxSteps",30U);
591.         Math::Interval<double> rainElevationRange=cfg.retrieveValue<Math::Interval<double>
     >("./rainElevationRange",Math::Interval<double>(-1000.0,1000.0));
592.         rainStrength=cfg.retrieveValue<GLfloat>("./rainStrength",0.25f);
593.         double evaporationRate=cfg.retrieveValue<double>("./evaporationRate",0.0);
594.         float demDistScale=cfg.retrieveValue<float>("./demDistScale",1.0f);
595.         std::string controlPipeName=cfg.retrieveString("./controlPipeName","");
596.
597.         /* Process command line parameters: */
598.         bool printHelp=false;
599.         const char* frameFilePrefix=0;
600.         const char* kinectServerName=0;
601.         bool useRemoteServer=false;
602.         int remoteServerPortId=26000;
603.         int windowIndex=0;
604.         renderSettings.push_back(RenderSettings());
```

```
605.    for(int i=1;i<argc;++i)
606.        {
607.        if(argv[i][0]=='-')
608.            {
609.            if(strcasecmp(argv[i]+1,"h")==0)
610.                printHelp=true;
611.            else if(strcasecmp(argv[i]+1,"remote")==0)
612.                {
613.                /* Check if there is an optional port number: */
614.                if(i+1<argc&&argv[i+1][0]>='0'&&argv[i+1][0]<='9')
615.                    {
616.                    ++i;
617.                    remoteServerPortId=atoi(argv[i]);
618.                    }
619.
620.                useRemoteServer=true;
621.                }
622.            else if(strcasecmp(argv[i]+1,"c")==0)
623.                {
624.                ++i;
625.                cameraIndex=atoi(argv[i]);
626.                }
627.            else if(strcasecmp(argv[i]+1,"f")==0)
628.                {
629.                ++i;
630.                frameFilePrefix=argv[i];
631.                }
632.            else if(strcasecmp(argv[i]+1,"p")==0)
633.                {
634.                ++i;
635.                kinectServerName=argv[i];
636.                }
637.            else if(strcasecmp(argv[i]+1,"s")==0)
638.                {
639.                ++i;
640.                scale=atof(argv[i]);
641.                }
642.            else if(strcasecmp(argv[i]+1,"slf")==0)
643.                {
644.                ++i;
645.                sandboxLayoutFileName=argv[i];
646.                }
647.            else if(strcasecmp(argv[i]+1,"er")==0)
648.                {
649.                ++i;
650.                double elevationMin=atof(argv[i]);
651.                ++i;
652.                double elevationMax=atof(argv[i]);
653.                elevationRange=Math::Interval<double>(elevationMin,elevationMax);
654.                }
655.            else if(strcasecmp(argv[i]+1,"hmp")==0)
656.                {
657.                /* Read height mapping plane coefficients: */
658.                haveHeightMapPlane=true;
659.                double hmp[4];
```

```
660.                    for(int j=0;j<4;++j)
661.                        {
662.                        ++i;
663.                        hmp[j]=atof(argv[i]);
664.                        }
665.                    heightMapPlane=Plane(Plane::Vector(hmp),hmp[3]);
666.                    heightMapPlane.normalize();
667.                    }
668.                else if(strcasecmp(argv[i]+1,"nas")==0)
669.                    {
670.                    ++i;
671.                    numAveragingSlots=atoi(argv[i]);
672.                    }
673.                else if(strcasecmp(argv[i]+1,"sp")==0)
674.                    {
675.                    ++i;
676.                    minNumSamples=atoi(argv[i]);
677.                    ++i;
678.                    maxVariance=atoi(argv[i]);
679.                    }
680.                else if(strcasecmp(argv[i]+1,"he")==0)
681.                    {
682.                    ++i;
683.                    hysteresis=float(atof(argv[i]));
684.                    }
685.                else if(strcasecmp(argv[i]+1,"wts")==0)
686.                    {
687.                    for(int j=0;j<2;++j)
688.                        {
689.                        ++i;
690.                        wtSize[j]=(unsigned int)(atoi(argv[i]));
691.                        }
692.                    }
693.                else if(strcasecmp(argv[i]+1,"ws")==0)
694.                    {
695.                    ++i;
696.                    waterSpeed=atof(argv[i]);
697.                    ++i;
698.                    waterMaxSteps=atoi(argv[i]);
699.                    }
700.                else if(strcasecmp(argv[i]+1,"rer")==0)
701.                    {
702.                    ++i;
703.                    double rainElevationMin=atof(argv[i]);
704.                    ++i;
705.                    double rainElevationMax=atof(argv[i]);
706.                    rainElevationRange=Math::Interval<double>(rainElevationMin,rainEleva
    tionMax);
707.                    }
708.                else if(strcasecmp(argv[i]+1,"rs")==0)
709.                    {
710.                    ++i;
711.                    rainStrength=GLfloat(atof(argv[i]));
712.                    }
713.                else if(strcasecmp(argv[i]+1,"evr")==0)
```

```
714.                        {
715.                        ++i;
716.                        evaporationRate=atof(argv[i]);
717.                        }
718.                  else if(strcasecmp(argv[i]+1,"dds")==0)
719.                        {
720.                        ++i;
721.                        demDistScale=float(atof(argv[i]));
722.                        }
723.                  else if(strcasecmp(argv[i]+1,"wi")==0)
724.                        {
725.                        ++i;
726.                        windowIndex=atoi(argv[i]);
727.
728.                        /* Extend the list of render settings if an index beyond the end is selected:
      */
729.                        while(int(renderSettings.size())<=windowIndex)
730.                              renderSettings.push_back(renderSettings.back());
731.
732.                        /* Disable fixed projector view on the new render settings: */
733.                        renderSettings.back().fixProjectorView=false;
734.                        }
735.                  else if(strcasecmp(argv[i]+1,"fpv")==0)
736.                        {
737.                        renderSettings.back().fixProjectorView=true;
738.                        if(i+1<argc&&argv[i+1][0]!='-')
739.                              {
740.                              /* Load the projector transformation file specified in the next
     argument: */
741.                              ++i;
742.                              renderSettings.back().loadProjectorTransform(argv[i]);
743.                              }
744.                        }
745.                  else if(strcasecmp(argv[i]+1,"nhs")==0)
746.                        renderSettings.back().hillshade=false;
747.                  else if(strcasecmp(argv[i]+1,"uhs")==0)
748.                        renderSettings.back().hillshade=true;
749.                  else if(strcasecmp(argv[i]+1,"ns")==0)
750.                        renderSettings.back().useShadows=false;
751.                  else if(strcasecmp(argv[i]+1,"us")==0)
752.                        renderSettings.back().useShadows=true;
753.                  else if(strcasecmp(argv[i]+1,"nhm")==0)
754.                        {
755.                        delete renderSettings.back().elevationColorMap;
756.                        renderSettings.back().elevationColorMap=0;
757.                        }
758.                  else if(strcasecmp(argv[i]+1,"uhm")==0)
759.                        {
760.                        if(i+1<argc&&argv[i+1][0]!='-')
761.                              {
762.                              /* Load the height color map file specified in the next argument: */
763.                              ++i;
764.                              renderSettings.back().loadHeightMap(argv[i]);
765.                              }
766.                        else
```

```
767.                                {
768.                                /* Load the default height color map: */
769.                                renderSettings.back().loadHeightMap(CONFIG_DEFAULTHEIGH
    TCOLORMAPFILENAME);
770.                                }
771.                        }
772.                else if(strcasecmp(argv[i]+1,"ncl")==0)
773.                        renderSettings.back().useContourLines=false;
774.                else if(strcasecmp(argv[i]+1,"ucl")==0)
775.                        {
776.                        renderSettings.back().useContourLines=true;
777.                        if(i+1<argc&&argv[i+1][0]!='-')
778.                                {
779.                                /* Read the contour line spacing: */
780.                                ++i;
781.                                renderSettings.back().contourLineSpacing=GLfloat(atof(argv[i]));
782.                                }
783.                        }
784.                else if(strcasecmp(argv[i]+1,"rws")==0)
785.                        renderSettings.back().renderWaterSurface=true;
786.                else if(strcasecmp(argv[i]+1,"rwt")==0)
787.                        renderSettings.back().renderWaterSurface=false;
788.                else if(strcasecmp(argv[i]+1,"wo")==0)
789.                        {
790.                        ++i;
791.                        renderSettings.back().waterOpacity=GLfloat(atof(argv[i]));
792.                        }
793.                else if(strcasecmp(argv[i]+1,"cp")==0)
794.                        {
795.                        ++i;
796.                        controlPipeName=argv[i];
797.                        }
798.                else
799.                        std::cerr<<"Ignoring unrecognized command line switch
    "<<argv[i]<<std::endl;
800.                }
801.        }
802.
803.     /* Print usage help if requested: */
804.     if(printHelp)
805.             printUsage();
806.
807.     if(frameFilePrefix!=0)
808.             {
809.             /* Open the selected pre-recorded 3D video files: */
810.             std::string colorFileName=frameFilePrefix;
811.             colorFileName.append(".color");
812.             std::string depthFileName=frameFilePrefix;
813.             depthFileName.append(".depth");
814.             camera=new Kinect::FileFrameSource(IO::openFile(colorFileName.c_str()),IO::ope
    nFile(depthFileName.c_str()));
815.             }
816.     else if(kinectServerName!=0)
817.             {
818.             /* Split the server name into host name and port: */
```

```
819.            const char* colonPtr=0;
820.            for(const char* snPtr=kinectServerName;*snPtr!='\0';++snPtr)
821.                 if(*snPtr==':')
822.                      colonPtr=snPtr;
823.            std::string hostName;
824.            int port;
825.            if(colonPtr!=0)
826.                 {
827.                 /* Extract host name and port: */
828.                 hostName=std::string(kinectServerName,colonPtr);
829.                 port=atoi(colonPtr+1);
830.                 }
831.            else
832.                 {
833.                 /* Use complete host name and default port: */
834.                 hostName=kinectServerName;
835.                 port=26000;
836.                 }
837.
838.            /* Open a multiplexed frame source for the given server host name and port
     number: */
839.            Kinect::MultiplexedFrameSource* source=Kinect::MultiplexedFrameSource::create
     (Comm::openTCPPipe(hostName.c_str(),port));
840.
841.            /* Use the server's first component stream as the camera device: */
842.            camera=source->getStream(0);
843.            }
844.       else
845.            {
846.            /* Open the 3D camera device of the selected index: */
847.            Kinect::DirectFrameSource* realCamera=Kinect::openDirectFrameSource(cameraI
     ndex,false);
848.            Misc::ConfigurationFileSection cameraConfigurationSection=cfg.getSection(camer
     aConfiguration.c_str());
849.            realCamera->configure(cameraConfigurationSection);
850.            camera=realCamera;
851.            }
852.       for(int i=0;i<2;++i)
853.            frameSize[i]=camera->getActualFrameSize(Kinect::FrameSource::DEPTH)[i];
854.
855.       /* Get the camera's per-pixel depth correction parameters and evaluate it on the depth
     frame's pixel grid: */
856.       Kinect::FrameSource::DepthCorrection* depthCorrection=camera-
     >getDepthCorrectionParameters();
857.       if(depthCorrection!=0)
858.            {
859.            pixelDepthCorrection=depthCorrection->getPixelCorrection(frameSize);
860.            delete depthCorrection;
861.            }
862.       else
863.            {
864.            /* Create dummy per-pixel depth correction parameters: */
865.            pixelDepthCorrection=new PixelDepthCorrection[frameSize[1]*frameSize[0]];
866.            PixelDepthCorrection* pdcPtr=pixelDepthCorrection;
867.            for(unsigned int y=0;y<frameSize[1];++y)
```

```
868.                    for(unsigned int x=0;x<frameSize[0];++x,++pdcPtr)
869.                        {
870.                        pdcPtr->scale=1.0f;
871.                        pdcPtr->offset=0.0f;
872.                        }
873.                }
874.
875.        /* Get the camera's intrinsic parameters: */
876.        cameraIps=camera->getIntrinsicParameters();
877.
878.        /* Read the sandbox layout file: */
879.        Geometry::Plane<double,3> basePlane;
880.        Geometry::Point<double,3> basePlaneCorners[4];
881.        {
882.        IO::ValueSource layoutSource(IO::openFile(sandboxLayoutFileName.c_str()));
883.        layoutSource.skipWs();
884.
885.        /* Read the base plane equation: */
886.        std::string s=layoutSource.readLine();
887.        basePlane=Misc::ValueCoder<Geometry::Plane<double,3> >::decode(s.c_str(),s.c_str()+
    s.length());
888.        basePlane.normalize();
889.
890.        /* Read the corners of the base quadrilateral and project them into the base plane: */
891.        for(int i=0;i<4;++i)
892.                {
893.                layoutSource.skipWs();
894.                s=layoutSource.readLine();
895.                basePlaneCorners[i]=basePlane.project(Misc::ValueCoder<Geometry::Point<doubl
    e,3> >::decode(s.c_str(),s.c_str()+s.length()));
896.                }
897.        }
898.
899.        /* Limit the valid elevation range to the intersection of the extents of all height color
    maps: */
900.        for(std::vector<RenderSettings>::iterator rsIt=renderSettings.begin();rsIt!=renderSettings
    .end();++rsIt)
901.                if(rsIt->elevationColorMap!=0)
902.                        {
903.                        Math::Interval<double> mapRange(rsIt->elevationColorMap-
    >getScalarRangeMin(),rsIt->elevationColorMap->getScalarRangeMax());
904.                        elevationRange.intersectInterval(mapRange);
905.                        }
906.
907.        /* Scale all sizes by the given scale factor: */
908.        double sf=scale/100.0; // Scale factor from cm to final units
909.        for(int i=0;i<3;++i)
910.                for(int j=0;j<4;++j)
911.                        cameraIps.depthProjection.getMatrix()(i,j)*=sf;
912.        basePlane=Geometry::Plane<double,3>(basePlane.getNormal(),basePlane.getOffset()*sf
    );
913.        for(int i=0;i<4;++i)
914.                for(int j=0;j<3;++j)
915.                        basePlaneCorners[i][j]*=sf;
916.        if(elevationRange!=Math::Interval<double>::full)
```

```
917.            elevationRange*=sf;
918.        if(rainElevationRange!=Math::Interval<double>::full)
919.            rainElevationRange*=sf;
920.        for(std::vector<RenderSettings>::iterator rsIt=renderSettings.begin();rsIt!=renderSettings
    .end();++rsIt)
921.            {
922.            if(rsIt->elevationColorMap!=0)
923.                rsIt->elevationColorMap->setScalarRange(rsIt->elevationColorMap-
    >getScalarRangeMin()*sf,rsIt->elevationColorMap->getScalarRangeMax()*sf);
924.            rsIt->contourLineSpacing*=sf;
925.            rsIt->waterOpacity/=sf;
926.            for(int i=0;i<4;++i)
927.                rsIt->projectorTransform.getMatrix()(i,3)*=sf;
928.            }
929.        rainStrength*=sf;
930.        evaporationRate*=sf;
931.        demDistScale*=sf;
932.
933.        /* Create the frame filter object: */
934.        frameFilter=new FrameFilter(frameSize,numAveragingSlots,pixelDepthCorrection,came
    raIps.depthProjection,basePlane);
935.        frameFilter-
    >setValidElevationInterval(cameraIps.depthProjection,basePlane,elevationRange.getMin(),ele
    vationRange.getMax());
936.        frameFilter->setStableParameters(minNumSamples,maxVariance);
937.        frameFilter->setHysteresis(hysteresis);
938.        frameFilter->setSpatialFilter(true);
939.        frameFilter-
    >setOutputFrameFunction(Misc::createFunctionCall(this,&Sandbox::receiveFilteredFrame));
940.
941.        if(waterSpeed>0.0)
942.            {
943.            /* Create the hand extractor object: */
944.            handExtractor=new HandExtractor(frameSize,pixelDepthCorrection,cameraIps.dept
    hProjection);
945.            }
946.
947.        /* Start streaming depth frames: */
948.        camera-
    >startStreaming(0,Misc::createFunctionCall(this,&Sandbox::rawDepthFrameDispatcher));
949.
950.        /* Create the depth image renderer: */
951.        depthImageRenderer=new DepthImageRenderer(frameSize);
952.        depthImageRenderer->setIntrinsics(cameraIps);
953.        depthImageRenderer->setBasePlane(basePlane);
954.
955.        {
956.        /* Calculate the transformation from camera space to sandbox space: */
957.        ONTransform::Vector z=basePlane.getNormal();
958.        ONTransform::Vector x=(basePlaneCorners[1]-
    basePlaneCorners[0])+(basePlaneCorners[3]-basePlaneCorners[2]);
959.        ONTransform::Vector y=z^x;
960.        boxTransform=ONTransform::rotate(Geometry::invert(ONTransform::Rotation::fromBa
    seVectors(x,y)));
```

```
961.        ONTransform::Point center=Geometry::mid(Geometry::mid(basePlaneCorners[0],basePl
    aneCorners[1]),Geometry::mid(basePlaneCorners[2],basePlaneCorners[3]));
962.        boxTransform*=ONTransform::translateToOriginFrom(center);
963.
964.        /* Calculate the size of the sandbox area: */
965.        boxSize=Geometry::dist(center,basePlaneCorners[0]);
966.        for(int i=1;i<4;++i)
967.              boxSize=Math::max(boxSize,Geometry::dist(center,basePlaneCorners[i]));
968.        }
969.
970.        /* Calculate a bounding box around all potential surfaces: */
971.        bbox=Box::empty;
972.        for(int i=0;i<4;++i)
973.              {
974.              bbox.addPoint(basePlaneCorners[i]+basePlane.getNormal()*elevationRange.getMi
    n());
975.              bbox.addPoint(basePlaneCorners[i]+basePlane.getNormal()*elevationRange.getMa
    x());
976.              }
977.
978.        if(waterSpeed>0.0)
979.              {
980.              /* Initialize the water flow simulator: */
981.              waterTable=new WaterTable2(wtSize[0],wtSize[1],depthImageRenderer,basePlane
    Corners);
982.              waterTable-
    >setElevationRange(elevationRange.getMin(),rainElevationRange.getMax());
983.              waterTable->setWaterDeposit(evaporationRate);
984.
985.              /* Register a render function with the water table: */
986.              addWaterFunction=Misc::createFunctionCall(this,&Sandbox::addWater);
987.              waterTable->addRenderFunction(addWaterFunction);
988.              addWaterFunctionRegistered=true;
989.              }
990.
991.        if(useRemoteServer)
992.              {
993.              /* Create a remote server: */
994.              try
995.                    {
996.                    remoteServer=new RemoteServer(this,remoteServerPortId,1.0/30.0);
997.                    }
998.              catch(const std::runtime_error& err)
999.                    {
1000.                         Misc::formattedConsoleError("Sandbox: Unable to create remote
    server on port %d due to exception %s",remoteServerPortId,err.what());
1001.                         }
1002.                   }
1003.
1004.             /* Initialize all surface renderers: */
1005.             for(std::vector<RenderSettings>::iterator rsIt=renderSettings.begin();rsIt!=render
    Settings.end();++rsIt)
1006.                   {
1007.                   /* Calculate the texture mapping plane for this renderer's height map: */
1008.                   if(rsIt->elevationColorMap!=0)
```

```
1009.                         {
1010.                         if(haveHeightMapPlane)
1011.                               rsIt->elevationColorMap->calcTexturePlane(heightMapPlane);
1012.                         else
1013.                               rsIt->elevationColorMap-
    >calcTexturePlane(depthImageRenderer);
1014.                         }
1015.
1016.                   /* Initialize the surface renderer: */
1017.                   rsIt->surfaceRenderer=new SurfaceRenderer(depthImageRenderer);
1018.                   rsIt->surfaceRenderer->setDrawContourLines(rsIt->useContourLines);
1019.                   rsIt->surfaceRenderer->setContourLineDistance(rsIt-
    >contourLineSpacing);
1020.                   rsIt->surfaceRenderer->setElevationColorMap(rsIt->elevationColorMap);
1021.                   rsIt->surfaceRenderer->setIlluminate(rsIt->hillshade);
1022.                   if(waterTable!=0)
1023.                         {
1024.                         if(rsIt->renderWaterSurface)
1025.                               {
1026.                               /* Create a water renderer: */
1027.                               rsIt->waterRenderer=new WaterRenderer(waterTable);
1028.                               }
1029.                         else
1030.                               {
1031.                               rsIt->surfaceRenderer->setWaterTable(waterTable);
1032.                               rsIt->surfaceRenderer->setAdvectWaterTexture(true);
1033.                               rsIt->surfaceRenderer->setWaterOpacity(rsIt->waterOpacity);
1034.                               }
1035.                         }
1036.                   rsIt->surfaceRenderer->setDemDistScale(demDistScale);
1037.                   }
1038.
1039.             #if 0
1040.             /* Create a fixed-position light source: */
1041.             sun=Vrui::getLightsourceManager()->createLightsource(true);
1042.             for(int i=0;i<Vrui::getNumViewers();++i)
1043.                   Vrui::getViewer(i)->setHeadlightState(false);
1044.             sun->enable();
1045.             sun->getLight().position=GLLight::Position(1,0,1,0);
1046.             #endif
1047.
1048.             /* Create the GUI: */
1049.             mainMenu=createMainMenu();
1050.             Vrui::setMainMenu(mainMenu);
1051.             if(waterTable!=0)
1052.                   waterControlDialog=createWaterControlDialog();
1053.
1054.             /* Initialize the custom tool classes: */
1055.             GlobalWaterTool::initClass(*Vrui::getToolManager());
1056.             LocalWaterTool::initClass(*Vrui::getToolManager());
1057.             DEMTool::initClass(*Vrui::getToolManager());
1058.             if(waterTable!=0)
1059.                   BathymetrySaverTool::initClass(waterTable,*Vrui::getToolManager());
1060.             addEventTool("Pause Topography",0,0);
1061.
```

```
1062.                 if(!controlPipeName.empty())
1063.                       {
1064.                       /* Open the control pipe in non-blocking mode: */
1065.                       controlPipeFd=open(controlPipeName.c_str(),O_RDONLY|O_NONBLOC
      K);
1066.                       if(controlPipeFd<0)
1067.                             std::cerr<<"Unable to open control pipe "<<controlPipeName<<";
      ignoring"<<std::endl;
1068.                       }
1069.
1070.                 /* Inhibit the screen saver: */
1071.                 Vrui::inhibitScreenSaver();
1072.
1073.                 /* Set the linear unit to support proper scaling: */
1074.                 Vrui::getCoordinateManager()-
      >setUnit(Geometry::LinearUnit(Geometry::LinearUnit::METER,scale/100.0));
1075.                 }
1076.
1077.           Sandbox::~Sandbox(void)
1078.                 {
1079.                 /* Stop streaming depth frames: */
1080.                 camera->stopStreaming();
1081.                 delete camera;
1082.                 delete frameFilter;
1083.
1084.                 /* Delete helper objects: */
1085.                 delete waterTable;
1086.                 delete depthImageRenderer;
1087.                 delete handExtractor;
1088.                 delete addWaterFunction;
1089.                 delete[] pixelDepthCorrection;
1090.                 delete remoteServer;
1091.
1092.                 delete mainMenu;
1093.                 delete waterControlDialog;
1094.
1095.                 close(controlPipeFd);
1096.                 }
1097.
1098.           void Sandbox::toolDestructionCallback(Vrui::ToolManager::ToolDestructionCallbac
      kData* cbData)
1099.                 {
1100.                 /* Check if the destroyed tool is the active DEM tool: */
1101.                 if(activeDem==dynamic_cast<DEM*>(cbData->tool))
1102.                       {
1103.                       /* Deactivate the active DEM tool: */
1104.                       activeDem=0;
1105.                       }
1106.                 }
1107.
1108.           namespace {
1109.
1110.           /*****************
1111.           Helper functions:
1112.           *****************/
```

```
1113.
1114.       std::vector<std::string> tokenizeLine(const char*& buffer)
1115.           {
1116.           std::vector<std::string> result;
1117.
1118.           /* Skip initial whitespace but not end-of-line: */
1119.           const char* bPtr=buffer;
1120.           while(*bPtr!='\0'&&*bPtr!='\n'&&isspace(*bPtr))
1121.               ++bPtr;
1122.
1123.           /* Extract white-space separated tokens until a newline or end-of-string are
    encountered: */
1124.           while(*bPtr!='\0'&&*bPtr!='\n')
1125.               {
1126.               /* Remember the start of the current token: */
1127.               const char* tokenStart=bPtr;
1128.
1129.               /* Find the end of the current token: */
1130.               while(*bPtr!='\0'&&!isspace(*bPtr))
1131.                   ++bPtr;
1132.
1133.               /* Extract the token: */
1134.               result.push_back(std::string(tokenStart,bPtr));
1135.
1136.               /* Skip whitespace but not end-of-line: */
1137.               while(*bPtr!='\0'&&*bPtr!='\n'&&isspace(*bPtr))
1138.                   ++bPtr;
1139.               }
1140.
1141.           /* Skip end-of-line: */
1142.           if(*bPtr=='\n')
1143.               ++bPtr;
1144.
1145.           /* Set the start of the next line and return the token list: */
1146.           buffer=bPtr;
1147.           return result;
1148.           }
1149.
1150.       bool isToken(const std::string& token,const char* pattern)
1151.           {
1152.           return strcasecmp(token.c_str(),pattern)==0;
1153.           }
1154.
1155.       }
1156.
1157.       void Sandbox::frame(void)
1158.           {
1159.           /* Call the remote server's frame method: */
1160.           if(remoteServer!=0)
1161.               remoteServer->frame(Vrui::getApplicationTime());
1162.
1163.           /* Check if the filtered frame has been updated: */
1164.           if(filteredFrames.lockNewValue())
1165.               {
1166.               /* Update the depth image renderer's depth image: */
```

```
1167.                     depthImageRenderer->setDepthImage(filteredFrames.getLockedValue());
1168.                     }
1169.
1170.                if(handExtractor!=0)
1171.                     {
1172.                     /* Lock the most recent extracted hand list: */
1173.                     handExtractor->lockNewExtractedHands();
1174.
1175.                     #if 0
1176.
1177.                     /* Register/unregister the rain rendering function based on whether hands
     have been detected: */
1178.                     bool registerWaterFunction=!handExtractor-
     >getLockedExtractedHands().empty();
1179.                     if(addWaterFunctionRegistered!=registerWaterFunction)
1180.                          {
1181.                          if(registerWaterFunction)
1182.                               waterTable->addRenderFunction(addWaterFunction);
1183.                          else
1184.                               waterTable->removeRenderFunction(addWaterFunction);
1185.                          addWaterFunctionRegistered=registerWaterFunction;
1186.                          }
1187.
1188.                     #endif
1189.                     }
1190.
1191.                /* Update all surface renderers: */
1192.                for(std::vector<RenderSettings>::iterator rsIt=renderSettings.begin();rsIt!=render
     Settings.end();++rsIt)
1193.                     rsIt->surfaceRenderer->setAnimationTime(Vrui::getApplicationTime());
1194.
1195.                /* Check if there is a control command on the control pipe: */
1196.                if(controlPipeFd>=0)
1197.                     {
1198.                     /* Try reading a chunk of data (will fail with EAGAIN if no data due to
     non-blocking access): */
1199.                     char commandBuffer[1024];
1200.                     ssize_t
     readResult=read(controlPipeFd,commandBuffer,sizeof(commandBuffer)-1);
1201.                     if(readResult>0)
1202.                          {
1203.                          commandBuffer[readResult]='\0';
1204.
1205.                          /* Extract commands line-by-line: */
1206.                          const char* cPtr=commandBuffer;
1207.                          while(*cPtr!='\0')
1208.                               {
1209.                               /* Split the current line into tokens and skip empty lines: */
1210.                               std::vector<std::string> tokens=tokenizeLine(cPtr);
1211.                               if(tokens.empty())
1212.                                    continue;
1213.
1214.                               /* Parse the command: */
1215.                               if(isToken(tokens[0],"waterSpeed"))
1216.                                    {
```

```
1217.                                   if(tokens.size()==2)
1218.                                       {
1219.                                       waterSpeed=atof(tokens[1].c_str());
1220.                                       if(waterSpeedSlider!=0)
1221.                                           waterSpeedSlider->setValue(waterSpeed);
1222.                                       }
1223.                                   else
1224.                                       std::cerr<<"Wrong number of arguments for
    waterSpeed control pipe command"<<std::endl;
1225.                                   }
1226.                               else if(isToken(tokens[0],"waterMaxSteps"))
1227.                                   {
1228.                                   if(tokens.size()==2)
1229.                                       {
1230.                                       waterMaxSteps=atoi(tokens[1].c_str());
1231.                                       if(waterMaxStepsSlider!=0)
1232.                                           waterMaxStepsSlider->setValue(waterMaxSteps);
1233.                                       }
1234.                                   else
1235.                                       std::cerr<<"Wrong number of arguments for
    waterMaxSteps control pipe command"<<std::endl;
1236.                                   }
1237.                               else if(isToken(tokens[0],"waterAttenuation"))
1238.                                   {
1239.                                   if(tokens.size()==2)
1240.                                       {
1241.                                       double attenuation=atof(tokens[1].c_str());
1242.                                       if(waterTable!=0)
1243.                                           waterTable->setAttenuation(GLfloat(1.0-
    attenuation));
1244.                                       if(waterAttenuationSlider!=0)
1245.                                           waterAttenuationSlider->setValue(attenuation);
1246.                                       }
1247.                                   else
1248.                                       std::cerr<<"Wrong number of arguments for
    waterAttenuation control pipe command"<<std::endl;
1249.                                   }
1250.                               else if(isToken(tokens[0],"colorMap"))
1251.                                   {
1252.                                   if(tokens.size()==2)
1253.                                       {
1254.                                       try
1255.                                           {
1256.                                           /* Update all height color maps: */
1257.                                           for(std::vector<RenderSettings>::iterator rsIt=rend
    erSettings.begin();rsIt!=renderSettings.end();++rsIt)
1258.                                               if(rsIt->elevationColorMap!=0)
1259.                                                   rsIt->elevationColorMap-
    >load(tokens[1].c_str());
1260.                                           }
1261.                                       catch(const std::runtime_error& err)
1262.                                           {
1263.                                           std::cerr<<"Cannot read height color map
    "<<tokens[1]<<" due to exception "<<err.what()<<std::endl;
1264.                                           }
```

```
1265.                                         }
1266.                                 else
1267.                                         std::cerr<<"Wrong number of arguments for colorMap
    control pipe command"<<std::endl;
1268.                                 }
1269.                         else if(isToken(tokens[0],"heightMapPlane"))
1270.                                 {
1271.                                 if(tokens.size()==5)
1272.                                         {
1273.                                         /* Read the height map plane equation: */
1274.                                         double hmp[4];
1275.                                         for(int i=0;i<4;++i)
1276.                                                 hmp[i]=atof(tokens[1+i].c_str());
1277.                                         Plane
    heightMapPlane=Plane(Plane::Vector(hmp),hmp[3]);
1278.                                         heightMapPlane.normalize();
1279.
1280.                                         /* Override the height mapping planes of all elevation
    color maps: */
1281.                                         for(std::vector<RenderSettings>::iterator rsIt=renderSet
    tings.begin();rsIt!=renderSettings.end();++rsIt)
1282.                                                 if(rsIt->elevationColorMap!=0)
1283.                                                         rsIt->elevationColorMap-
    >calcTexturePlane(heightMapPlane);
1284.                                         }
1285.                                 else
1286.                                         std::cerr<<"Wrong number of arguments for
    heightMapPlane control pipe command"<<std::endl;
1287.                                 }
1288.                         else if(isToken(tokens[0],"useContourLines"))
1289.                                 {
1290.                                 if(tokens.size()==2)
1291.                                         {
1292.                                         /* Parse the command parameter: */
1293.                                         if(isToken(tokens[1],"on")||isToken(tokens[1],"off"))
1294.                                                 {
1295.                                                 /* Enable or disable contour lines on all surface
    renderers: */
1296.                                                 bool useContourLines=isToken(tokens[1],"on");
1297.                                                 for(std::vector<RenderSettings>::iterator rsIt=rend
    erSettings.begin();rsIt!=renderSettings.end();++rsIt)
1298.                                                         rsIt->surfaceRenderer-
    >setDrawContourLines(useContourLines);
1299.                                                 }
1300.                                         else
1301.                                                 std::cerr<<"Invalid parameter "<<tokens[1]<<" for
    useContourLines control pipe command"<<std::endl;
1302.                                         }
1303.                                 else
1304.                                         std::cerr<<"Wrong number of arguments for
    contourLineSpacing control pipe command"<<std::endl;
1305.                                 }
1306.                         else if(isToken(tokens[0],"contourLineSpacing"))
1307.                                 {
1308.                                 if(tokens.size()==2)
```

```
1309.                                       {
1310.                                       /* Parse the contour line distance: */
1311.                                       GLfloat
    contourLineSpacing=GLfloat(atof(tokens[1].c_str()));
1312.
1313.                                       /* Check if the requested spacing is valid: */
1314.                                       if(contourLineSpacing>0.0f)
1315.                                           {
1316.                                           /* Override the contour line spacing of all surface
    renderers: */
1317.                                           for(std::vector<RenderSettings>::iterator rsIt=rend
    erSettings.begin();rsIt!=renderSettings.end();++rsIt)
1318.                                               rsIt->surfaceRenderer-
    >setContourLineDistance(contourLineSpacing);
1319.                                           }
1320.                                       else
1321.                                           std::cerr<<"Invalid parameter
    "<<contourLineSpacing<<" for contourLineSpacing control pipe command"<<std::endl;
1322.                                       }
1323.                                   else
1324.                                       std::cerr<<"Wrong number of arguments for
    contourLineSpacing control pipe command"<<std::endl;
1325.                                   }
1326.                               else if(isToken(tokens[0],"dippingBed"))
1327.                                   {
1328.                                   if(tokens.size()==2&&isToken(tokens[1],"off"))
1329.                                       {
1330.                                       /* Disable dipping bed rendering on all surface
    renderers: */
1331.                                       for(std::vector<RenderSettings>::iterator rsIt=renderSet
    tings.begin();rsIt!=renderSettings.end();++rsIt)
1332.                                           rsIt->surfaceRenderer-
    >setDrawDippingBed(false);
1333.                                       }
1334.                                   else if(tokens.size()==5)
1335.                                       {
1336.                                       /* Read the dipping bed plane equation: */
1337.                                       GLfloat dbp[4];
1338.                                       for(int i=0;i<4;++i)
1339.                                           dbp[i]=GLfloat(atof(tokens[1+i].c_str()));
1340.                                       SurfaceRenderer::Plane dippingBedPlane=SurfaceRend
    erer::Plane(SurfaceRenderer::Plane::Vector(dbp),dbp[3]);
1341.                                       dippingBedPlane.normalize();
1342.
1343.                                       /* Enable dipping bed rendering and set the dipping bed
    plane equation on all surface renderers: */
1344.                                       for(std::vector<RenderSettings>::iterator rsIt=renderSet
    tings.begin();rsIt!=renderSettings.end();++rsIt)
1345.                                           {
1346.                                           rsIt->surfaceRenderer->setDrawDippingBed(true);
1347.                                           rsIt->surfaceRenderer-
    >setDippingBedPlane(dippingBedPlane);
1348.                                           }
1349.                                       }
1350.                                   else
```

```
1351.                                         std::cerr<<"Wrong number of arguments for
    dippingBed control pipe command"<<std::endl;
1352.                                 }
1353.                             else if(isToken(tokens[0],"foldedDippingBed"))
1354.                                 {
1355.                                 if(tokens.size()==6)
1356.                                     {
1357.                                     /* Read the dipping bed coefficients: */
1358.                                     GLfloat dbc[5];
1359.                                     for(int i=0;i<5;++i)
1360.                                         dbc[i]=GLfloat(atof(tokens[1+i].c_str()));
1361.
1362.                                     /* Enable dipping bed rendering and set the dipping bed
    coefficients on all surface renderers: */
1363.                                     for(std::vector<RenderSettings>::iterator rsIt=renderSet
    tings.begin();rsIt!=renderSettings.end();++rsIt)
1364.                                         {
1365.                                         rsIt->surfaceRenderer->setDrawDippingBed(true);
1366.                                         rsIt->surfaceRenderer-
    >setDippingBedCoeffs(dbc);
1367.                                         }
1368.                                     }
1369.                                 else
1370.                                     std::cerr<<"Wrong number of arguments for
    foldedDippingBed control pipe command"<<std::endl;
1371.                                 }
1372.                             else if(isToken(tokens[0],"dippingBedThickness"))
1373.                                 {
1374.                                 if(tokens.size()==2)
1375.                                     {
1376.                                     /* Read the dipping bed thickness: */
1377.                                     float dippingBedThickness=float(atof(tokens[1].c_str())
    );
1378.
1379.                                     /* Set the dipping bed thickness on all surface
    renderers: */
1380.                                     for(std::vector<RenderSettings>::iterator rsIt=renderSet
    tings.begin();rsIt!=renderSettings.end();++rsIt)
1381.                                         rsIt->surfaceRenderer-
    >setDippingBedThickness(dippingBedThickness);
1382.                                     }
1383.                                 else
1384.                                     std::cerr<<"Wrong number of arguments for
    dippingBedThickness control pipe command"<<std::endl;
1385.                                 }
1386.                             else
1387.                                 std::cerr<<"Unrecognized control pipe command
    "<<tokens[0]<<std::endl;
1388.                             }
1389.                         }
1390.                     }
1391.
1392.                 if(frameRateTextField!=0&&Vrui::getWidgetManager()-
    >isVisible(waterControlDialog))
1393.                     {
```

```
1394.                     /* Update the frame rate display: */
1395.                     frameRateTextField->setValue(1.0/Vrui::getCurrentFrameTime());
1396.                     }
1397.
1398.             if(pauseUpdates)
1399.                     Vrui::scheduleUpdate(Vrui::getApplicationTime()+1.0/30.0);
1400.             }
1401.
1402.         void Sandbox::display(GLContextData& contextData) const
1403.             {
1404.             /* Get the data item: */
1405.             DataItem* dataItem=contextData.retrieveDataItem<DataItem>(this);
1406.
1407.             /* Get the rendering settings for this window: */
1408.             const Vrui::DisplayState& ds=Vrui::getDisplayState(contextData);
1409.             const Vrui::VRWindow* window=ds.window;
1410.             int windowIndex;
1411.             for(windowIndex=0;windowIndex<Vrui::getNumWindows()&&window!=Vrui::
    getWindow(windowIndex);++windowIndex)
1412.                     ;
1413.             const RenderSettings& rs=windowIndex<int(renderSettings.size())?renderSettin
    gs[windowIndex]:renderSettings.back();
1414.
1415.             /* Check if the water simulation state needs to be updated: */
1416.             if(waterTable!=0&&dataItem->waterTableTime!=Vrui::getApplicationTime())
1417.                     {
1418.                     /* Retrieve a potential pending grid read-back request: */
1419.                     GridRequest::Request request=gridRequest.getRequest();
1420.
1421.                     /* Update the water table's bathymetry grid: */
1422.                     waterTable->updateBathymetry(contextData);
1423.
1424.                     /* Check if the grid request is active and wants bathymetry data: */
1425.                     if(request.isActive()&&request.bathymetryBuffer!=0)
1426.                             {
1427.                             /* Read back the current bathymetry grid: */
1428.                             waterTable->bindBathymetryTexture(contextData);
1429.                             glGetTexImage(GL_TEXTURE_RECTANGLE_ARB,0,GL_RED,GL
    _FLOAT,request.bathymetryBuffer);
1430.                             glBindTexture(GL_TEXTURE_RECTANGLE_ARB,0);
1431.                             }
1432.
1433.                     /* Run the water flow simulation's main pass: */
1434.                     GLfloat totalTimeStep=GLfloat(Vrui::getFrameTime()*waterSpeed);
1435.                     unsigned int numSteps=0;
1436.                     while(numSteps<waterMaxSteps-1U&&totalTimeStep>1.0e-8f)
1437.                             {
1438.                             /* Run with a self-determined time step to maintain stability: */
1439.                             waterTable->setMaxStepSize(totalTimeStep);
1440.                             GLfloat timeStep=waterTable->runSimulationStep(false,contextData);
1441.                             totalTimeStep-=timeStep;
1442.                             ++numSteps;
1443.                             }
1444.                     #if 0
1445.                     if(totalTimeStep>1.0e-8f)
```

```
1446.                         {
1447.                         std::cout<<'.'<<std::flush;
1448.                         /* Force the final step to avoid simulation slow-down: */
1449.                         waterTable->setMaxStepSize(totalTimeStep);
1450.                         GLfloat timeStep=waterTable->runSimulationStep(true,contextData);
1451.                         totalTimeStep-=timeStep;
1452.                         ++numSteps;
1453.                         }
1454.                     #else
1455.                     if(totalTimeStep>1.0e-8f)
1456.                         std::cout<<"Ran out of time by "<<totalTimeStep<<std::endl;
1457.                     #endif
1458.
1459.                     /* Check if the grid request is active and wants water level data: */
1460.                     if(request.isActive()&&request.waterLevelBuffer!=0)
1461.                         {
1462.                         /* Read back the current water level grid: */
1463.                         waterTable->bindQuantityTexture(contextData);
1464.                         glGetTexImage(GL_TEXTURE_RECTANGLE_ARB,0,GL_RED,GL
    _FLOAT,request.waterLevelBuffer);
1465.                         glBindTexture(GL_TEXTURE_RECTANGLE_ARB,0);
1466.                         }
1467.
1468.                     /* Finish an active grid request: */
1469.                     if(request.isActive())
1470.                         request.complete();
1471.
1472.                     /* Mark the water simulation state as up-to-date for this frame: */
1473.                     dataItem->waterTableTime=Vrui::getApplicationTime();
1474.                     }
1475.
1476.                 /* Calculate the projection matrix: */
1477.                 PTransform projection=ds.projection;
1478.                 if(rs.fixProjectorView&&rs.projectorTransformValid)
1479.                     {
1480.                     /* Use the projector transformation instead: */
1481.                     projection=rs.projectorTransform;
1482.
1483.                     /* Multiply with the inverse modelview transformation so that lighting still
    works as usual: */
1484.                     projection*=Geometry::invert(ds.modelviewNavigational);
1485.                     }
1486.
1487.                 if(rs.hillshade)
1488.                     {
1489.                     /* Set the surface material: */
1490.                     glMaterial(GLMaterialEnums::FRONT,rs.surfaceMaterial);
1491.                     }
1492.
1493.                 #if 0
1494.                 if(rs.hillshade&&rs.useShadows)
1495.                     {
1496.                     /* Set up OpenGL state: */
1497.                     glPushAttrib(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT|GL
    _ENABLE_BIT|GL_POLYGON_BIT);
```

```
1498.
1499.                   GLLightTracker& lt=*contextData.getLightTracker();
1500.
1501.                   /* Save the currently-bound frame buffer and viewport: */
1502.                   GLint currentFrameBuffer;
1503.                   glGetIntegerv(GL_FRAMEBUFFER_BINDING_EXT,&currentFrameBuff
    er);
1504.                   GLint currentViewport[4];
1505.                   glGetIntegerv(GL_VIEWPORT,currentViewport);
1506.
1507.                   /*****************************************************
    *******
1508.                   First rendering pass: Global ambient illumination only
1509.                   *****************************************************
    ******/
1510.
1511.                   /* Draw the surface mesh: */
1512.                   surfaceRenderer->glRenderGlobalAmbientHeightMap(dataItem-
    >heightColorMapObject,contextData);
1513.
1514.                   /*****************************************************
    *******
1515.                   Second rendering pass: Add local illumination for every light source
1516.                   *****************************************************
    ******/
1517.
1518.                   /* Enable additive rendering: */
1519.                   glEnable(GL_BLEND);
1520.                   glBlendFunc(GL_ONE,GL_ONE);
1521.                   glDepthFunc(GL_LEQUAL);
1522.                   glDepthMask(GL_FALSE);
1523.
1524.                   for(int lightSourceIndex=0;lightSourceIndex<lt.getMaxNumLights();++ligh
    tSourceIndex)
1525.                         if(lt.getLightState(lightSourceIndex).isEnabled())
1526.                               {
1527.                               /*****************************************************
    ************
1528.                               First step: Render to the light source's shadow map
1529.                               *****************************************************
    **********/
1530.
1531.                               /* Set up OpenGL state to render to the shadow map: */
1532.                               glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,dataItem-
    >shadowFramebufferObject);
1533.                               glViewport(0,0,dataItem->shadowBufferSize[0],dataItem-
    >shadowBufferSize[1]);
1534.                               glDepthMask(GL_TRUE);
1535.                               glClear(GL_DEPTH_BUFFER_BIT);
1536.                               glCullFace(GL_FRONT);
1537.
1538.                               /*****************************************************
    *********
1539.                               Calculate the shadow projection matrix:
```

```
                    ****************************************************
*******/

                    /* Get the light source position in eye space: */
                    Geometry::HVector<float,3> lightPosEc;
                    glGetLightfv(GL_LIGHT0+lightSourceIndex,GL_POSITION,lightPosEc.getComponents());

                    /* Transform the light source position to camera space: */
                    Vrui::ONTransform::HVector lightPosCc=Vrui::getDisplayState(contextData).modelviewNavigational.inverseTransform(Vrui::ONTransform::HVector(lightPosEc));

                    /* Calculate the direction vector from the center of the bounding box to the light source: */
                    Point bboxCenter=Geometry::mid(bbox.min,bbox.max);
                    Vrui::Vector lightDirCc=Vrui::Vector(lightPosCc.getComponents())-Vrui::Vector(bboxCenter.getComponents())*lightPosCc[3];

                    /* Build a transformation that aligns the light direction with the positive z axis: */
                    Vrui::ONTransform shadowModelview=Vrui::ONTransform::rotate(Vrui::Rotation::rotateFromTo(lightDirCc,Vrui::Vector(0,0,1)));
                    shadowModelview*=Vrui::ONTransform::translateToOriginFrom(bboxCenter);

                    /* Create a projection matrix, based on whether the light is positional or directional: */
                    PTransform shadowProjection(0.0);
                    if(lightPosEc[3]!=0.0f)
                        {
                        /* Modify the modelview transformation such that the light source is at the origin: */
                        shadowModelview.leftMultiply(Vrui::ONTransform::translate(Vrui::Vector(0,0,-lightDirCc.mag())));

                        /*********************************************
***********
                        Create a perspective projection:
                        ************************************************
**********/

                        /* Calculate the perspective bounding box of the surface bounding box in eye space: */
                        Box pBox=Box::empty;
                        for(int i=0;i<8;++i)
                            {
                            Point bc=shadowModelview.transform(bbox.getVertex(i));
                            pBox.addPoint(Point(-bc[0]/bc[2],-bc[1]/bc[2],-bc[2]));
                            }

                        /* Upload the frustum matrix: */
                        double l=pBox.min[0]*pBox.min[2];
                        double r=pBox.max[0]*pBox.min[2];
```

```
1579.                                   double b=pBox.min[1]*pBox.min[2];
1580.                                   double t=pBox.max[1]*pBox.min[2];
1581.                                   double n=pBox.min[2];
1582.                                   double f=pBox.max[2];
1583.                                   shadowProjection.getMatrix()(0,0)=2.0*n/(r-l);
1584.                                   shadowProjection.getMatrix()(0,2)=(r+l)/(r-l);
1585.                                   shadowProjection.getMatrix()(1,1)=2.0*n/(t-b);
1586.                                   shadowProjection.getMatrix()(1,2)=(t+b)/(t-b);
1587.                                   shadowProjection.getMatrix()(2,2)=-(f+n)/(f-n);
1588.                                   shadowProjection.getMatrix()(2,3)=-2.0*f*n/(f-n);
1589.                                   shadowProjection.getMatrix()(3,2)=-1.0;
1590.                                   }
1591.                           else
1592.                                   {
1593.                                   /*************************************************
    ***********
1594.                                   Create a perspective projection:
1595.                                   *************************************************
    **********/
1596.
1597.                                   /* Transform the bounding box with the modelview
    transformation: */
1598.                                   Box bboxEc=bbox;
1599.                                   bboxEc.transform(shadowModelview);
1600.
1601.                                   /* Upload the ortho matrix: */
1602.                                   double l=bboxEc.min[0];
1603.                                   double r=bboxEc.max[0];
1604.                                   double b=bboxEc.min[1];
1605.                                   double t=bboxEc.max[1];
1606.                                   double n=-bboxEc.max[2];
1607.                                   double f=-bboxEc.min[2];
1608.                                   shadowProjection.getMatrix()(0,0)=2.0/(r-l);
1609.                                   shadowProjection.getMatrix()(0,3)=-(r+l)/(r-l);
1610.                                   shadowProjection.getMatrix()(1,1)=2.0/(t-b);
1611.                                   shadowProjection.getMatrix()(1,3)=-(t+b)/(t-b);
1612.                                   shadowProjection.getMatrix()(2,2)=-2.0/(f-n);
1613.                                   shadowProjection.getMatrix()(2,3)=-(f+n)/(f-n);
1614.                                   shadowProjection.getMatrix()(3,3)=1.0;
1615.                                   }
1616.
1617.                           /* Multiply the shadow modelview matrix onto the shadow
    projection matrix: */
1618.                           shadowProjection*=shadowModelview;
1619.
1620.                           /* Draw the surface into the shadow buffer: */
1621.                           surfaceRenderer-
    >glRenderDepthOnly(shadowProjection,contextData);
1622.
1623.                           /* Reset OpenGL state: */
1624.                           glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,currentFra
    meBuffer);
1625.                           glViewport(currentViewport[0],currentViewport[1],currentViewp
    ort[2],currentViewport[3]);
1626.                           glCullFace(GL_BACK);
```

```
1627.                                 glDepthMask(GL_FALSE);
1628.
1629.                                 #if SAVEDEPTH
1630.                                 /* Save the depth image: */
1631.                                 {
1632.                                 glBindTexture(GL_TEXTURE_2D,dataItem-
    >shadowDepthTextureObject);
1633.                                 GLfloat* depthTextureImage=new GLfloat[dataItem-
    >shadowBufferSize[1]*dataItem->shadowBufferSize[0]];
1634.                                 glGetTexImage(GL_TEXTURE_2D,0,GL_DEPTH_COMPONE
    NT,GL_FLOAT,depthTextureImage);
1635.                                 glBindTexture(GL_TEXTURE_2D,0);
1636.                                 Images::RGBImage dti(dataItem-
    >shadowBufferSize[0],dataItem->shadowBufferSize[1]);
1637.                                 GLfloat* dtiPtr=depthTextureImage;
1638.                                 Images::RGBImage::Color* ciPtr=dti.modifyPixels();
1639.                                 for(int y=0;y<dataItem->shadowBufferSize[1];++y)
1640.                                       for(int x=0;x<dataItem-
    >shadowBufferSize[0];++x,++dtiPtr,++ciPtr)
1641.                                             {
1642.                                             GLColor<GLfloat,3> tc(*dtiPtr,*dtiPtr,*dtiPtr);
1643.                                             *ciPtr=tc;
1644.                                             }
1645.                                 delete[] depthTextureImage;
1646.                                 Images::writeImageFile(dti,"DepthImage.png");
1647.                                 }
1648.                                 #endif
1649.
1650.                                 /* Draw the surface using the shadow texture: */
1651.                                 rs.surfaceRenderer-
    >glRenderShadowedIlluminatedHeightMap(dataItem->heightColorMapObject,dataItem-
    >shadowDepthTextureObject,shadowProjection,contextData);
1652.                                 }
1653.
1654.                           /* Reset OpenGL state: */
1655.                           glPopAttrib();
1656.                           }
1657.                     else
1658.                     #endif
1659.                           {
1660.                           /* Render the surface in a single pass: */
1661.                           rs.surfaceRenderer-
    >renderSinglePass(ds.viewport,projection,ds.modelviewNavigational,contextData);
1662.                           }
1663.
1664.                     if(rs.waterRenderer!=0)
1665.                           {
1666.                           /* Draw the water surface: */
1667.                           glMaterialAmbientAndDiffuse(GLMaterialEnums::FRONT,GLColor<GLfl
    oat,4>(0.0f,0.5f,0.8f));
1668.                           glMaterialSpecular(GLMaterialEnums::FRONT,GLColor<GLfloat,4>(1.0f,
    1.0f,1.0f));
1669.                           glMaterialShininess(GLMaterialEnums::FRONT,64.0f);
1670.                           glEnable(GL_BLEND);
1671.                           glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```

```
1672.                   rs.waterRenderer-
    >render(projection,ds.modelviewNavigational,contextData);
1673.                   glDisable(GL_BLEND);
1674.                   }
1675.
1676.           /* Call the remote server's render method: */
1677.           if(remoteServer!=0)
1678.                   {
1679.                   glMatrixMode(GL_PROJECTION);
1680.                   glPushMatrix();
1681.                   glLoadMatrix(projection);
1682.                   glMatrixMode(GL_MODELVIEW);
1683.                   remoteServer->glRenderAction(contextData);
1684.                   glMatrixMode(GL_PROJECTION);
1685.                   glPopMatrix();
1686.                   glMatrixMode(GL_MODELVIEW);
1687.                   }
1688.           }
1689.
1690.       void Sandbox::resetNavigation(void)
1691.           {
1692.           /* Construct a navigation transformation to center the sandbox area in the
    display, facing the viewer, with the long sandbox axis facing to the right: */
1693.           Vrui::NavTransform nav=Vrui::NavTransform::translateFromOriginTo(Vrui::get
    DisplayCenter());
1694.           nav*=Vrui::NavTransform::scale(Vrui::getDisplaySize()/boxSize);
1695.           Vrui::Vector y=Vrui::getUpDirection();
1696.           Vrui::Vector z=Vrui::getForwardDirection();
1697.           Vrui::Vector x=z^y;
1698.           nav*=Vrui::NavTransform::rotate(Vrui::Rotation::fromBaseVectors(x,y));
1699.           nav*=boxTransform;
1700.           Vrui::setNavigationTransformation(nav);
1701.           }
1702.
1703.       void Sandbox::eventCallback(Vrui::Application::EventID eventId,Vrui::InputDevice::
    ButtonCallbackData* cbData)
1704.           {
1705.           if(cbData->newButtonState)
1706.                   {
1707.                   switch(eventId)
1708.                           {
1709.                           case 0:
1710.                                   /* Invert the current pause setting: */
1711.                                   pauseUpdates=!pauseUpdates;
1712.
1713.                                   /* Update the main menu toggle: */
1714.                                   pauseUpdatesToggle->setToggle(pauseUpdates);
1715.
1716.                                   break;
1717.                           }
1718.                   }
1719.           }
1720.
1721.       void Sandbox::initContext(GLContextData& contextData) const
1722.           {
```

```
1723.               /* Create a data item and add it to the context: */
1724.               DataItem* dataItem=new DataItem;
1725.               contextData.addDataItem(this,dataItem);
1726.
1727.               {
1728.               /* Save the currently bound frame buffer: */
1729.               GLint currentFrameBuffer;
1730.               glGetIntegerv(GL_FRAMEBUFFER_BINDING_EXT,&currentFrameBuffer);
1731.
1732.               /* Set the default shadow buffer size: */
1733.               dataItem->shadowBufferSize[0]=1024;
1734.               dataItem->shadowBufferSize[1]=1024;
1735.
1736.               /* Generate the shadow rendering frame buffer: */
1737.               glGenFramebuffersEXT(1,&dataItem->shadowFramebufferObject);
1738.               glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,dataItem-
    >shadowFramebufferObject);
1739.
1740.               /* Generate a depth texture for shadow rendering: */
1741.               glGenTextures(1,&dataItem->shadowDepthTextureObject);
1742.               glBindTexture(GL_TEXTURE_2D,dataItem->shadowDepthTextureObject);
1743.               glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINE
    AR);
1744.               glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LIN
    EAR);
1745.               glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
1746.               glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
1747.               glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_COMPARE_MODE_AR
    B,GL_COMPARE_R_TO_TEXTURE);
1748.               glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_COMPARE_FUNC_AR
    B,GL_LEQUAL);
1749.               glTexParameteri(GL_TEXTURE_2D,GL_DEPTH_TEXTURE_MODE_ARB,G
    L_INTENSITY);
1750.               glTexImage2D(GL_TEXTURE_2D,0,GL_DEPTH_COMPONENT24_ARB,dat
    aItem->shadowBufferSize[0],dataItem-
    >shadowBufferSize[1],0,GL_DEPTH_COMPONENT,GL_UNSIGNED_BYTE,0);
1751.               glBindTexture(GL_TEXTURE_2D,0);
1752.
1753.               /* Attach the depth texture to the frame buffer object: */
1754.               glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,GL_DEPTH_ATTA
    CHMENT_EXT,GL_TEXTURE_2D,dataItem->shadowDepthTextureObject,0);
1755.               glDrawBuffer(GL_NONE);
1756.               glReadBuffer(GL_NONE);
1757.               glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,currentFrameBuffer);
1758.               }
1759.               }
1760.
1761.          VRUI_APPLICATION_RUN(Sandbox)
```

Anexo 4: Certificado de Traducción

FINE- NED ENGLISH
tßßOUßO£ IßfTIT¥T£
Líderes en la Enseñanza del Inglés

Ing. María Belén Novillo Sánchez.
ENGLISB TEACHER - FINE TUNED ENGLISH CIA **LTDA.**

CERTIFICA:

Que el documento aquí compuesto es fiel traduccíón del idioma español a1 idioma inglés del trabajo de titulación **°ESTUDIO** E **IMPLEMENTACIOŃ DE SIMULADOR DE MAPA TOPOGRÁFICO INTERACTIVO PARA LA FACULTAD DE ENERGÍA** DE **LA UNL"** autoría de Carlos Fabricio Condolo Castillo con nùmero de cédula 1105267734, estudiante de la carrera de Ingeniería en Electrónica y Telecomunícaciones de la Universidad Nacional de Loja.

Lo certifico en honor a la verdad y autorizo a la interesada hacer uso del presente en lo que a sus intereses convenga.

Loja, 14 de julio del 2022

Ing. María Belén Novillo Sánchez.
ENGLISH **TEACHER- FINE TUNED ENGLISH CIA LTDA.**

FINE-TUNED ENGLISH LANGUAGE INSTITUTE
Líderes en la Enseñanza del Inglés