“DESARROLLO DE UNA HERRAMIENTA DE SOFTWARE BAJO LINUX PARA LA MODELIZACIÓN, ANIMACIÓN Y VISUALIZACIÓN DE ESCENAS EN TRES DIMENSIONES ESPECIFICADAS CON LA INTERFAZ PIXAR RENDERMAN”

Víctor Javier Cerquera Parra

Código 9621930

Alejandro Gutiérrez Muñoz

Código 9616665

Universidad del Valle

Facultad de Ingeniería

Escuela de Ingeniería de Sistemas y Computación

Santiago de Cali

2001

“DESARROLLO DE UNA HERRAMIENTA DE SOFTWARE BAJO LINUX PARA LA MODELIZACIÓN, ANIMACIÓN Y VISUALIZACIÓN DE ESCENAS EN TRES DIMENSIONES ESPECIFICADAS CON LA INTERFAZ PIXAR RENDERMAN”

Víctor Javier Cerquera Parra
Alejandro Gutiérrez Muñoz

Trabajo de grado presentado como requisito para optar al título de Ingeniero de Sistemas

Director
Mauricio Gaona Cuevas Ph.D.
Director del Programa Académico de Ingeniería de Sistemas

Universidad del Valle
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Santiago de Cali
2001

Nota de Aceptación

Director

Jurado

Jurado

Santiago de Cali, 11 de septiembre de 2001

A mi Madre, por la confianza depositada, el apoyo brindado en el momento oportuno, y la valentía que ha demostrado poseer a lo largo de su vida.

Víctor Javier Cerquera

Al Espíritu Santo por haberme dado fuerza y sabiduría en un largo camino de continuo aprendizaje.

Alejandro Gutiérrez

AGRADECIMIENTOS

Alejandro expresa sus agradecimientos a:

Mis padres por su apoyo, a todos mis compañeros de universidad por su invaluable ayuda en momentos decisivos, al profesor Héctor Angulo por su gran colaboración, aporte intelectual y personal a lo largo de toda mi carrera, a Javier, Claudia, Mauricio, Consuelo, Alex, Angela, Maria C, Leonardo, Miltón, Sandra, Said, Alvaro y muy especialmente a Edna.

Víctor Javier expresa sus agradecimientos a:

Mi madre por su fe y confianza, a Claudia, Consuelo y Milton por su apoyo en momentos difíciles y a mis compañeros por su amistad a lo largo de cuatro años.

Los autores expresan sus agradecimientos:

Al profesor Mauricio Gaona, Director del programa académico de Ingeniería de Sistemas por su orientación y ayuda.
Al profesor José María Bañón por la enseñanza recibida durante el curso de computación gráfica.
A los profesores de la Escuela por la enseñanza de los fundamentos necesarios para llevar a cabo esta empresa.

TABLA DE CONTENIDO

0. Introducción 9

1. Objetivos 11

2. Formulación del Problema 13

3. Antecedentes 14

4. Marco Teórico 20

4.1. La interfaz Renderman 17

4.1.1. Descripción General 20

4.1.1.1. Una Interfaz para la descripción de escenas 3-D 20

4.1.1.2. Lenguaje de Sombreado De Renderman (Shading Language) 23

4.1.1.3. Características y Capacidades de Renderman 24

4.1.1.3.1. Características Requeridas para la generación de escenas fotorealistas 24

4.1.1.3.2. Capacidades Avanzadas para la generación de escenas fotorealistas 26

4.1.2. Descripción de Escenas fotorealistas usando la Interfaz Renderman 27

4.1.2.1. Enlace con Lenguaje C 28

4.1.2.2. Generación por medio de archivos con formato RIB 28

4.1.2.3. Generación de escenas en la práctica 29

4.1.3. Estado de los Gráficos 36

4.1.3.1. Opciones (Options) 41

4.1.3.1.1. Cámara (camera) 41

4.1.3.1.2. Despliegue (Display) 48

4.1.3.2. Atributos (Attributes) 51

4.1.4. Transformaciones 59

4.1.5. Primitivas Geométricas 61
4.2. Blue Moon Renderign Tools (BMRT) 69
4.3. OpenGL 70
4.3.1. OpenGL como una máquina de estados 71
4.3.2. Librerías 72
4.3.3. La guía auxiliar de programación de la librería OpenGL 72
4.3.4. Dibujando Objetos 3D 74
4.3.5. Descripción de puntos, líneas y polígonos 74
4.3.5.1. ¿ Cómo especificar los vértices de un objeto ? 76
4.3.5.2. Dibujo de Primitivas Geométricas 76
4.3.5.3. Restricciones al Usar glBegin() y glEnd() 77
4.3.5.4. Dibujo de puntos, líneas y polígonos 78
4.3.5.5. Vectores Normales 82
4.4. Lenguaje Unificado de Modelado "UML". 82
4.4.1. Inicio de UML 83
4.4.2. Objetivo de UML 83
4.4.3. Modelado de objetos 84
4.4.4. Artefactos para el Desarrollo de Proyectos 84
4.4.4.1. Diagramas de Casos de Uso 85
4.4.4.2. Diagramas de Clases 85
4.4.4.3. Diagramas de secuencia 90
4.4.4.4. Diagramas de colaboración 91
4.4.4.5. Diagramas de actividad 92
4.4.4.6. Diagramas de estado 92
4.4.4.7. Diagramas de Implementación 93
4.4.4.7.1. Diagramas de componentes 93
4.4.4.7.2. Diagrama de plataformas o despliegue 93

4.5. Ingeniería del software 94
4.5.1. Paradigmas del Ciclo de Vida de Desarrollo de Software 95
4.5.2. Paradigmas para desarrollo de software 98
5. Fase de Análisis 100
5.1. Requerimientos del Sistema 100
5.1.1. Requerimientos Funcionales 100
5.1.2. Requerimientos no Funcionales 105
5.2. Definición de los Casos de Uso 106
5.2.1. Diagrama de Casos de Uso 106
5.2.1.1. Diagramas de Casos de Uso Extendido 107
5.2.2. Documentación de los Casos de Uso 111
5.3. Modelo Conceptual 152
6. Diseño del Sistema 153
6.1. Diagramas de Secuencia 153
6.2. Diagrama de Estructura Estática 187
6.2.1. Diagrama de Estructura Estática (Detalle) 188
6.3. Diagramas de Colaboración 197
6.4. Contratos 203
7. Construcción de la herramienta de Software 227
8. Conclusiones 239
9. Recomendaciones 241
10. Bibliografía 242

0. INTRODUCCIÓN

A mediados de la década de los 80's en el medio de la industria cinematográfica y del mundo de la informática comenzó a nacer una técnica de realización de efectos visuales conocida como animación por computador; ésta consistía en generar escenas, es decir imágenes o secuencias de ellas, utilizando los recursos computacionales para esto.

Dentro del campo de la animación por computador destacó por su novedad y por la amplía gamma de posibilidades que ofrecía para la industria la animación tridimensional, donde las escenas generadas presentaban características visuales que desplegaban las imágenes con volumen y profundidad basándose en complejas técnicas de generación de imágenes las cuales producían como resultado una imagen con aspecto de tres dimensiones en el ambiente bidimensional de la pantalla de la computadora.

Con el transcurrir de los años este campo de los efectos visuales ha crecido a pasos agigantados, a tal punto que hoy en día las escenas generadas por computador en ocasiones se confunden con facilidad con fotografías o grabaciones en vídeo del mundo real.

Uno de los principales inconvenientes que han tenido las técnicas de animación tridimensionales son los altos recursos computacionales que requieren para su elaboración, ya que durante el proceso de generación de la imagen los algoritmos encargados de darle el aspecto tridimensional a ésta realizan una gran cantidad de cálculos matemáticos que consumen demasiados recursos computacionales, a tal punto que para la generación de

imágenes hoy en día se utilizan supercomputadoras con una gran cantidad de procesadores en paralelo dedicadas única y exclusivamente a esta labor.

La exigencia de la industria hoy en día recae sobre las imágenes con aspecto fotorealista, las cuales a su vez son las que mayores recursos requieren para su generación; estas imágenes son generadas a partir de algoritmos basados en la técnica de radiosidad, la cual es una de las dos clases de técnicas para la generación de imágenes (la otra es el trazador de rayos).

El presente trabajo de grado consiste en el desarrollo de una herramienta de software bajo Linux para la modelización, animación y visualización de escenas en tres dimensiones especificadas con la interfaz Pixar Renderman, más conocida como RIB (Renderman Interface Bytestream), para posteriormente utilizar el algoritmo de generación de imágenes BMRT (Blue Moon Rendering Tools) el cual recibe escenas definidas en el estándar RIB y genera imágenes de calidad fotorealista basado en algoritmos de trazador de rayos y radiosidad. Además de permitir realizar en red el proceso de generación de la imagen; este algoritmo es gratuito para propósitos educacionales y está disponible en distintas plataformas, incluyendo Linux.

1. OBJETIVOS

1.1. Objetivo General

Desarrollar una herramienta de software que funcione bajo el sistema operativo Linux para la modelización, animación y visualización de escenas en tres dimensiones especificadas con la interfaz Pixar Renderman.

1.2. Objetivos Específicos

1. Efectuar el análisis y diseño de la aplicación usando una metodología orientada a objetos.

2. Utilizar el lenguaje de modelado UML para modelar la aplicación.

3. Realizar un estudio y análisis completo de la interfaz Pixar Rederman.

4. Diseñar e implementar una interfaz gráfica de usuario (GUI) en c++ mediante el API (Aplication Programing Interface) QT que permita el acceso a los módulos de modelización, animación, visualización y herramientas externas del programa.

5. Desarrollar un módulo de modelización de escenas en tres dimensiones utilizando las librerías OpenGL que implemente la mayoría de las primitivas básicas que permite OpenGL (esferas, polígonos, líneas, etc) y sus correspondientes operaciones de modificación (escalamiento, rotación,

traslación, etc) y permita almacenar y cargar la escena modelada en el formato RIB (Renderman Interface Bytestream).

6. Desarrollar un módulo de animación que permita realizar modificaciones a los objetos de la escena en un intervalo de tiempo determinado y que almacene la información de cada frame (cuadro de la escena) en el archivo RIB (Renderman Interface Bytestream) correspondiente a la escena.

7. Desarrollar un módulo de visualización en OpenGL que le permita al usuario ver de manera rápida el resultado de la animación sin necesidad de realizar el proceso de generación de las imágenes.

8. Desarrollar un módulo de generación de imágenes en red haciendo uso de la herramienta Farm de BMRT (Blue Moon Rendering Tools) que permita generar una escena en diferentes computadoras que tengan instalado el algoritmo BMRT.

9. Desarrollar un módulo de creación y modificación de materiales el cual permita la definición de los mismos según los estándares de Pixar Renderman y la carga de materiales ya creados.

10. Desarrollar un módulo que permita la generación final de escena utilizando el algoritmo BMRT (Blue Moon Rendering Tools).

2. FORMULACIÓN DEL PROBLEMA

En la actualidad se encuentran diferentes aplicaciones de software que nos permiten realizar escenas tridimensionales generadas por computador, la gran mayoría de estas aplicaciones son comerciales y con altos costos en el mercado, aunque también se encuentran aplicaciones gratuitas como shareware (software compartido) o versiones beta o de prueba las cuales deshabilitan algunas de las principales opciones del software como grabar, o generan imágenes con letreros publicitarios.

Al hablar del campo de la generación de imágenes con aspecto fotorealista es más notoria la ausencia de software gratuito para realizar dichas imágenes, a tal punto que sólo como distribución gratuita (para uso educativo) se conocen en el medio 2 programas K-3D y AC3D y open source (código libre o abierto) tan sólo uno llamado Mops.

El problema entonces radica en la poca cantidad de programas totalmente gratuitos y que además permitan al usuario modificar, si así lo desea, su código para adaptarlo más a sus necesidades y a la vez ampliar su funcionalidad de tal manera que los demás usuarios de este programa disfruten los cambios realizados por otros.

Por lo tanto se plantea: Desarrollar una herramienta de software bajo el sistema operativo Linux, en la modalidad open source, para la modelización, animación y visualización de escenas en tres dimensiones especificadas con la interfaz Pixar Renderman o RIB para posteriormente realizar la generación de la imagen con el algoritmo de generación de imágenes BMRT (Blue Moon Rendering Tools).

3. ANTECEDENTES

En el campo de la generación de imágenes con características fotorealistas se ha destacado desde sus comienzos la empresa Pixar, conocida en el medio como los creadores de grandes producciones como Toy Story y participando de los efectos especiales de un sinnúmero de producciones de ciencia ficción.

La empresa Pixar encabezada por su fundador John Lassetter, el cual ha sido ganador en varias ocasiones de premios de la academia, ha sido la abanderada del desarrollo de nuevas técnicas y algoritmos para la generación de imágenes; pero sin duda alguna su mayor contribución al medio la realizó a comienzos de la década pasada al proponer la interfaz Renderman, la cual es una interfaz estándar entre programas de modelización y programas de generación de imágenes (Programas de Renderización) capaces de producir imágenes con calidad fotorealista.

En el transcurso de la década pasada se desarrollaron en torno a la interfaz Pixar Renderman numerosos programas de renderización capaces de producir imágenes de calidad fotorealista que utilizaban la definición del formato RIB (Renderman Interface Bytestream); dentro de éstos programas se encuentran los siguientes:

- **Pixar's Renderman Toolkit (PRMan)**: este es la implementación de renderman más ampliamente usada, y de hecho domina la escena de los renderizadores para películas de alto presupuesto. PRMan (Photorealistic Renderman) ha sido usado por la mayoría de casas de producción, incluyendo ILM (Industrial Light and Magic), Digital Domain, Disney, Sony

Pictures Imagesworks, Tippet y otros. Este fue usado para los efectos de render para las producciones: The Abyss, Terminator 2, Jurassic Park, Casper, Apollo 13, Contact, Starship Troopers, Toy Story (Classic y II), A Bug's Life, Star Wars I, Dinosaur, entre muchas otras. Este programa corre bajo plataformas SGI (Silicon Graphics), Sun, DEC Alpha OSF-1, Linux/Intel, Linux/Alpha, y Windows NT/Intel, se vende en el mercado por US$5.000 por CPU.

- **The Blue Moon Rendering Tools (BMRT):** este paquete está muy cerca de ser el segundo mejor programa de renderización después de PRMan de Pixar. BMRT soporta trazador de rayos y radiosidad, áreas de iluminación, implementación completa del lenguaje de sombreado, volúmenes e imágenes sombreadas, desplazamientos, y otras características avanzadas. Este también viene con un previsualizador en tiempo real de escenas definidas en formato RIB utilizando OpenGL. BMRT corre sobre SGI, Linux, Solaris, FreeBSD, Windows 95/98/NT/Me/2000, DEC (OSF y Alpha Linux). Cabe destacar el hecho que este es totalmente gratis para uso no comercial y el costo para usos comerciales es extremadamente económico (alrededor de los US$80), este implementa casi por completo la interfaz Renderman y ha sido usado en varias producciones incluyendo A Bug's Life, Stuart Little, Gone in 60 Seconds, y Hollow Man.

- **RenderDotC**: este es un renderizador de escaneo de línea que soporta la interfaz Renderman producido por Dot C Software, este corre bajo Windows, SGI, y Linux. Sus autores tienen un gran conocimiento sobre Renderman y se muestra como un gran paquete.

- **The University of Erlangen's Vision system**: es compatible con la interfaz Renderman.
- **GMAN**: este es un proyecto de interfaz gráfica de usuario (GUI) compatible con la interfaz Renderman, por el momento sólo existen algunas porciones de código y aún no trabaja como renderizador.
- **Advanced Rendering Technology**: este es distribuido por RenderDrive, es un hardware motor de trazador de rayos que es compatible con Renderman (sólo en la geometría).
- **AQSIS**: (se pronuncia "axis") está aún en versión Alpha y es compatible con la interfaz Renderman.
- **Siren**: es otro renderizador compatible con Renderman, escrito por Scott Iverson. Este corre sobre DOS, y de acuerdo a esta descripción implementa mucho de la especificación de Renderman 3.1. Su versión en Windows se llama AIR.
- **3Dlight**: es un renderizador compatible con Renderman y con la arquitectura REYES (Render Everything You Ever See). Esta disponible para IRIX, Linux y Windows.

Existen también programas encargados de la modelización de escenas tridimensionales y animación de las mismas que permite almacenarlas en el estándar Renderman tales como:

- **SolidThinking**: distribuido por GESTEL, corre bajo Windows NT y 95, soporta formato RIB.

- **Rhino3D:** es un modelador basado en NURBS que corre en plataformas Windows y recientemente adicionó el soporte para RIB.
- **Maya/Alias Wavefront:** este es uno de los más populares paquetes de animación de alta calidad soporta también el formato RIB.
- **3D Studio Max**: es el programa de autodesk 3D para la modelización y animación de escenas tridimensionales, aunque como tal no exporta formatos RIB si existen extensiones desarrolladas por terceros para esta labor.
- **Houdini:** es el software de la casa Side Effects Software realizadores de PRIMS otro software para modelización tridimensional, es uno de los más populares en el mercado y soporta el formato RIB.
- **SoftImage:** es sin duda alguna uno de los mejores software de animación que existen y soporta el formato RIB.
- **LightWave 3D:** al igual que 3D Studio Max no soporta nativamente el formato RIB, pero existen pluggins encargados de convertirlo.
- **Hash Animation Master:** existe un pluggin para que soporte formato RIB.
- **XFROG:** es un paquete de modelización orgánica, es decir, de plantas y árboles, puede generar formato RIB y corre bajo SGI y Linux.
- **Sced:** es un modelador CSG basado en restricciones que puede generar formato RIB, escrito por Stephen Chenney.
- **ShellyLib2.0:** es un generador de formas por línea de comandos que puede generar formato RIB.

- **Font3D:** este programa puede tomar texto ASCII y fuentes True Type y convertirlos en letras 3D para posteriormente grabarlas en formato RIB.
- **Geometique:** es un modelador de división de superficies que puede exportar a formato RIB. Corre bajo Windows NT.
- **Partsyt:** es una librería para el desarrollo de sistemas de partículas escrito por Jonathan Merritt, el cual genera escenas compatibles con Renderman.
- **Reptile Labour Project:** esta empresa desarrolló un sistema de partículas llamado flow que genera salida en formato RIB.
- **Okino Computer Graphics, Inc:** esta empresa tiene 2 productos llamados 'NuGraf Rendering System' y 'PolyTrans', los cuales incluyen la capacidad de exportar a Renderman.
- **Poser 4:** es un modelador de formas y poses humanas que posee la capacidad de exportar a formato RIB.
- **PIXELS 3D:** es un programa que corre bajo Mac que tiene la capacidad de exportar a formato RIB.

Los programas anteriormente mencionados son todos comerciales y sólo algunos tienen disponible versiones de prueba. A continuación se muestran los únicos programas que son gratuitos para la modelización y animación de escenas en 3D:

- **AC3D:** es un programa basado en la modelización de polígonos, el cual es gratuito o shareware (software compartido). Este programa permite

exportar a formato RIB, corre bajo Linux, Windows, SGI y Sun. Este software sólo se consigue en forma de binario ejecutable.

- **K-3D:** es un sistema de modelización y animación de GPL que puede exportar a formato RIB y corre bajo Linux y Windows, al igual que el anterior se distribuye en forma de binario ejecutable.

- **Mops:** Es un entorno de modelización que puede escribir a formato RIB y es el **único** software conocido que se distribuye como open source (código abierto). Corre bajo Linux, SGI y NT.

4. MARCO TEÓRICO

4.1. La interfaz Renderman

4.1.1 Descripción General

4.1.1.1 Una Interfaz para la descripción de escenas 3-D

La interfaz Renderman es una interfaz estándar entre programas de modelización capaces de producir imágenes de calidad fotorealista. Un programa de renderización que use la interfaz Renderman difiere en uno que no lo hace en:

- Un programa de renderización debe simular una cámara real y esto conlleva muchos atributos como la posición y la dirección de la vista. La alta calidad implica que la simulación no introduzca artefactos del proceso computacional.

  Expresado en la terminología de computación gráfica, esto significa que un programa de renderización fotorealista debe estar en capacidad de:

  - Remover los objetos que no deben aparecer en la imagen definida en la vista.

  - Filtrado espacial, es decir que los artefactos del proceso computacional tales como el aliasing (dentado) no están presentes.

- Difuminado de la imagen, así los artefactos pequeños, conocidos como quantos no aparecen en la imagen final.

- Filtrado temporal, esto es para el efecto de distorsión de movimiento o blur.

- Profundidad de campo, es decir que los objetos que se encuentran desenfocados deben aparecer borrosos.

- Un programa de renderización fotorealista debe también aceptar primitivas geométricas curvas, así no sólo pueden ser exactamente desplegadas sino que permiten que las formas básicas puedan incluirlas y ayudar a formas más naturales. Para esto se requiere de parches, cuádricas, y representación de sólidos, como bien la habilidad de tratar escenas complejas con una gran cantidad primitivas geométricas en el orden de 10.000 a 1.000.000.

- Un programa de renderización fotorealista debe estar en capacidad de simular las propiedades ópticas de los diferentes materiales y fuentes de luz. Esto incluye modelos de sombreado de superficies que describen cómo la luz interactúa con una superficie hecha de un material dado, modelos de sombreado volumétrico que describen cómo la luz es repartida en una región en el espacio, y modelos de fuentes de luz que describen el color y la intensidad de la luz emitida en diferentes direcciones. Lograr un mayor realismo a menudo requiere que las propiedades de superficie de un objeto varíen. Estas propiedades son

generalmente controladas por los mapas de texturas en las imágenes sobre una superficie. Los mapas de textura son usados de muchas maneras: directamente mapeados en la imagen para cambiar el color, mapeo de transparencia y de levantamiento para cambiar el vector normal, mapas de desplazamiento para modificar la posición, mapas de entorno y reflexión para calcular eficientemente la iluminación global, y mapas de sombras para simular la presencia de sombras.

La interfaz Renderman permite que la información necesaria para la descripción de una imagen fotorealista sea enviada a un programa de renderización de forma compacta y eficiente. La interfaz por si misma esta diseñada para manejar diferentes dispositivos de hardware, implementaciones de software y algoritmos de renderización. Muchos tipos de sistemas de renderización son soportados por esta interfaz incluyendo z-buffer, escaneo de líneas, trazador de rayos, generación de terrenos, generación de moléculas o esferas y la arquitectura de la renderización Reyes. Para lograr esto la interfaz no especifica cómo la imagen es generada sino cómo se desea. La interfaz está diseñada para soportar sistemas orientado a lotes y sistemas de renderización en tiempo real. La renderización en tiempo real está dada de tal forma que asegura que la información necesaria para dibujar una primitiva en particular está disponible cuando la primitiva es definida.

La interfaz Renderman no permite definir la totalidad de las descripciones de las escenas, esto debido a que no es posible dar soporte al gran número de descripciones de modelos que existen, o mejor, es muy complejo. Un claro ejemplo lo constituyen los modelos de color donde se encuentran un buen

número de los cuales sólo un subconjunto es soportado por Renderman. Otro ejemplo son las primitivas geométricas, las primitivas definidas por Renderman están definidas para ser primitivas de renderización, no de modelización. Además la tarea de convertir formas de modelización complejas a estructuras básicas es de los programas de modelización.

La interfaz Renderman no está diseñada para ser ambiente completo de programación tridimensional ya que no soporta cosas como: espacios bidimensionales, como textos o líneas 2-D y curvas; además se encuentra sin restricciones de sistemas de ventanas, dispositivos de entrada y eventos ya que estos no están considerados con la interfaz.

La interfaz Renderman es una colección de procedimientos para transferir la descripción de una escena a un programa de renderización. Un programa de renderización toma esta entrada y produce una imagen, esta imagen puede ser inmediatamente desplegada en pantalla o almacenada en un archivo. Los archivos de imágenes pueden ser utilizados como mapas de textura, pero Renderman no especifica un formato estándar para esto.

4.1.1.2. Lenguaje de Sombreado De Renderman (Shading Language)

Este es un lenguaje de programación para extender la funcionalidad predefinida de la interfaz Renderman. Se pueden crear nuevos materiales y fuentes de luz usando este lenguaje. Este lenguaje es también utilizado para especificar atenuaciones volumétricas, desplazamientos, y funciones simples

de procesamiento de imágenes. En este lenguaje está definido toda la funcionalidad requerida para los modelos de sombreado.

Un lenguaje de sombreado es una parte esencial de un programa de renderización de alta calidad, no se puede esperar que con simples ecuaciones de sombreado se produzcan resultados de aspecto fotorealista y muestre toda la complejidad de los posibles modelos de materiales, por esto se creó el Renderman Shading Language.

4.1.1.3. Características y Capacidades de Renderman

La interfaz Renderman fue diseñada con un estilo arriba-abajo, es decir pregunta primero que información necesita para especificar con suficiente detalle la escena para darle características fotorealistas; la generación de imágenes fotorealistas es el reto de los programas de renderización los cuales en la mayoría no implementan todas las características provistas por la interfaz Renderman. A continuación se definen cuales características son totalmente requeridas para alcanzar el cometido de generar imágenes con calidad fotorealista y cuales son consideradas avanzadas y logran mayor realismo en la escena, por lo que estas últimas son implementadas por pocos programas de renderización.

4.1.1.3.1. Características Requeridas para la generación de escenas fotorealistas

Todos los programas de renderización que usan la interfaz renderman deben ser especificados como se describe en The RenderMan Interface Version 3.2.

Las implementaciones que son librerías enlazables de C deben proveer como punto de entrada para todas las subrutinas y funciones los encabezados definidos en la librería ri.h. (provista por Pixar) Las implementaciones que se ejecuten de manera independiente (standalone), deben aceptar un archivo definido en el formato RIB (Renderman Interface Bytestream).

Todos los programas de renderización que implementen la interfaz Renderman deben:

- Proveer la jerarquía completa del estado de los gráficos, incluyendo los atributos y pilas de transformaciones y la lista de luces activas.

- Realizar transformaciones en vista ortogonal y perspectiva.

- Realizar difuminación y antialias de pixeles

- Realizar corrección gamma y difuminado después de la quantización

- Producir archivos de imágenes que contengan una combinación de RGB, Alpha y Z. Las resoluciones de esos archivos deben ser especificadas por el usuario.

- Proveer todas las primitivas geométricas especificadas en The RenderMan Interface Version 3.2. y todas las variaciones estándar aplicables a cada primitiva.

- Proveer la habilidad de realizar cálculos de sombreados usando el Renderman Shading Language (Lenguaje de Sombreado).

- Proveer la habilidad de indexar mapas de texturas, ambiente, profundidad y sombras.

- Proveer las fuentes de luz estándar, superficies, volúmenes, desplazamiento, y sombreado de imágenes requeridas por la especificación.

Los programas de renderización que usan la interfaz Renderman reciben todos los datos a través de la interfaz, y no deben contar con rutinas adicionales para proveer características de la escena.

4.1.1.3.2. Capacidades Avanzadas para la generación de escenas fotorealistas

Los programas de renderización deben también proveer de una o más de las siguientes capacidades avanzadas:

- Modelización de Sólidos: es la habilidad para definir modelos sólidos como un conjunto de superficies y combinarlas usando el conjunto de operaciones de intersección, unión y diferencia.
- Nivel de detalle: es la habilidad para especificar varias definiciones del mismo modelo y seleccionar uno basado en el tamaño de la pantalla estimado del modelo.

- Desenfoque de movimiento (Motion blur): es la habilidad para procesar primitivas en movimiento y aplicarles procesos de antialiasing a través del tiempo.
- Profundidad de campo: es la habilidad de simular enfoque a diferentes profundidades.
- Proyecciones especiales de cámara: es la habilidad de realizar proyecciones de cámara no estándares tales como proyecciones esféricas u omnimax.
- Desplazamientos: es la habilidad para manejar desplazamientos.
- Espectro de Colores: es la habilidad para calcular colores con un número arbitrario de muestras de color espectral.
- Sombreado de volúmenes: es la habilidad de añadir y procesar procedimientos de sombreado volumétrico.
- Trazador de rayos: es la habilidad de evaluar modelos de iluminación global usando trazador de rayos.
- Iluminación global: es la habilidad de evaluar modelos de iluminación indirecta usando radiosidad u otras técnicas de iluminación global.
- Áreas que se comportan como fuentes de luz: es la habilidad de iluminar superficies por medio de áreas que se comportan como fuentes de luz.

4.1.2. Descripción de Escenas fotorealistas usando la Interfaz Renderman

Ahora que se ha descrito qué es renderman y para qué sirve, se mostrará la manera en la cual se puede realizar la descripción de una escena utilizando los medios proveídos por la interfaz, estos son Enlace con lenguaje C, y el formato de archivo RIB (Renderman Interface Bytestream).

4.1.2.1 Enlace con Lenguaje C

Los procedimientos, primitivas, tokens, variables y demás que definen la interfaz Renderman se encuentran presentes por medio de una implementación en ANSI C, las declaraciones a dichas características de la interfaz se especifican en el archivo de encabezados de C ri.h, el cual debe ser incluido desde un archivo en lenguaje C y posteriormente proveer una librería de enlace al momento de la compilación en la cual se implementan dichas declaraciones. Por lo general el programa de generación de la imagen provee dicha librería de enlace, y quizá su propio archivo de encabezado ri.h que puede diferir un poco del proveído por Pixar.

Para propósitos didácticos en este informe se realizaran ejemplos utilizando el programa de generación de escenas BMRT (Blue Moon Rendering Tools) el cual provee su propia librería de enlace llamada libribout.lib, y su propio archivo de encabezado ri.h, además soporta la generación de escenas a partir de un archivo .rib, es decir definido con el formato RIB el cual explicaremos en la siguiente sección.

4.1.2.2. Generación por medio de archivos con formato RIB

El formato RIB permite tener la definición de la escena utilizando la interfaz Renderman. El formato RIB es en pocas palabras un archivo de texto plano que contiene la definición de la escena utilizando la sintaxis proveída por la interfaz para esto, dicho archivo sirve como de medio de almacenamiento de

escenas para ser posteriormente generadas pasándolo como entrada a un programa de generación de escenas (como BMRT).

4.1.2.3 Generación de escenas en la práctica

Un ejemplo simple de cómo funciona la descripción y generación de escenas utilizando el enlace con C y el formato RIB es el siguiente:

Archivo en formato RIB:

```
Format 300 300 1
ScreenWindow -2 2 -2 2

WorldBegin

  Translate 0 0 1

  TransformBegin
    Color [1 1 1]
    Sphere 3000 -3000 3000 360
  TransformEnd

 Color [0 0 1]

  TransformBegin
    Sphere 0.4 -0.4 0.4 360
  TransformEnd

  TransformBegin
    Translate -1.2 1.2 0
    Rotate 90 1 0 0
    Cylinder 0.4 -0.6 0.2 360
```

```
  TransformEnd

  TransformBegin
    Translate 0 1.2 0
    Disk 0 0.4 360
  TransformEnd

  TransformBegin
    Translate 1.2 0.9 0
    Rotate -90 1 0 0
    Cone 0.9 0.4 360
  TransformEnd

  TransformBegin
    Translate 1.2 0 0
    Torus 0.4 0.15 0 360 360
  TransformEnd

  TransformBegin
    Translate -1.2 0 0
    Rotate -90 1 0 0
    Hyperboloid -0.4 -0 -0.4   0 -0.4 0.4   360
  TransformEnd

  TransformBegin
    Translate 0 -1.6 0
    Rotate -90 1 0 0
    Paraboloid 0.4 0 0.8 360
  TransformEnd
WorldEnd
```

Este ejemplo muestra la creación de una escena en la cual se encuentran las primitivas básicas permitidas por Renderman, más adelante se explicará

como es la definición formal de las escenas basados en la sintaxis de la interfaz.

Para visualizar los resultados del ejemplo es necesario grabar el archivo con extensión .rib, por ejemplo primitivas.rib y posteriormente pasarlo como parámetro de entrada a un programa de renderización como BMRT.

En BMRT el comando encargado de realizar la generación de la imagen es rendrib, el cual recibe como parámetro el nombre del archivo .rib y las opciones de despliegue y generación. Para el ejemplo primitivas.rib el comando para generar la imagen sería:

$> rendrib –d primitivas.rib

la opción –d significa que despliegue la imagen mientras la genera. La ejecución de este comando con el archivo anterior generará una imagen de 300x300 pixeles con las primitivas básicas.

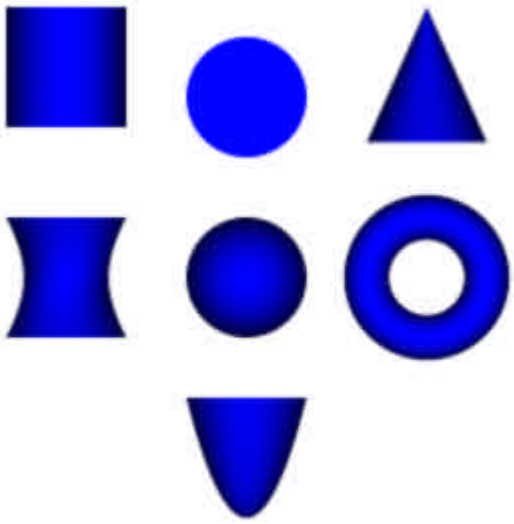


Figura 1: Ejemplo de primitivas descritas en RIB.

Este mismo ejemplo pero utilizando el enlace con el lenguaje C se vería de la siguiente manera:

Archivo en lenguaje C:

```
#include <stdlib.h>
#include <stdio.h>
#include <ri.h>

void main()
{

  RtColor blue = { 0.0f, 0.0f, 1.0f };
  RtColor white = { 1.0f, 1.0f, 1.0f };

     RiBegin("rendrib");

      RiDisplay("prueba.tif",RI_FRAMEBUFFER, RI_RGBA, RI_NULL);
```

```
RiTranslate(0.0, 0.0, 1.0);
RiFormat(300, 300, 1.0);
RiScreenWindow(-2, 2, -2, 2);

RiWorldBegin();


RiTransformBegin();
    RiColor(white);
    RiSphere(3000.0, -3000.0f, 3000.0, 360.0, RI_NULL);
RiTransformEnd();


  RiColor(blue);

RiTransformBegin();
    RiSphere(0.4, -0.4, 0.4, 360.0, RI_NULL);
RiTransformEnd();

RiTransformBegin();
    RiTranslate(-1.2, 1.2, 0);
    RiRotate(90, 1, 0, 0);
    RiCylinder(0.4, -0.6, 0.2, 360, RI_NULL);
RiTransformEnd();


RiTransformBegin();
    RiTranslate(0, 1.2, 0);
    RiDisk(0, 0.4, 360, RI_NULL);
RiTransformEnd();

RiTransformBegin();
    RiTranslate(1.2, 0.9, 0);
    RiRotate(-90, 1, 0, 0);
    RiCone(0.9, 0.4, 360, RI_NULL);
```

```
        RiTransformEnd();

        RiTransformBegin();
            RiTranslate(1.2, 0, 0);
            RiTorus(0.4, 0.15, 0, 360, 360, RI_NULL);
        RiTransformEnd();

        RtPoint a = { -0.4, -0.0, -0.4 };
        RtPoint b = { 0.0, -0.4, 0.4 };

        RiTransformBegin();
            RiTranslate(-1.2, 0, 0);
            RiRotate(-90, 1, 0, 0);
            RiHyperboloid(a, b, 360, RI_NULL);
        RiTransformEnd();

        RiTransformBegin();
            RiTranslate(0, -1.6, 0);
            RiRotate(-90, 1, 0, 0);
            RiParaboloid(0.4, 0.0, 0.8, 360, RI_NULL);
        RiTransformEnd();

      RiWorldEnd();

    RiEnd();

}
```

Este archivo genera la misma imagen que el ejemplo primitivas.rib es decir la figura 1, este archivo debe ser compilado haciendo inclusión a las bibliotecas de generación de rib y al archivo de encabezado ri.h, la línea de compilación sería como sigue:

```
$> g++ -I../include primitivas.c -L../lib -lribout -o pr
```

donde la bandera –I indica el path donde se encuentra el archivo ri.h y la bandera –L indica el path donde se encuentra la biblioteca de enlace libribout.lib.

Como se puede observar en los ejemplos las similitudes entre los dos estilos de generación son evidentes, e igualmente se puede observar que representa mayor facilidad de manejo el formato RIB, por lo que a través de este documento se hará referencia y presentaran ejemplos en este formato y no en el de enlace a C.

Es importante notar que los encabezados de las funciones y tokens definidos en el archivo ri.h referencia el formato y argumentos de entrada de estas por lo que si se desea comparar los argumentos de las funciones y los tipos definidos por la interfaz renderman se puede hacer revisando este archivo. (Para una mayor documentación ver el manual de referencia "The Renderman Interface" de Pixar)

No sobra una anotación para aclarar que las funciones y los tipos utilizados tanto en el formato RIB, como en el enlace con lenguaje C son idénticos, con la diferencia que en el enlace con C las funciones comienzan con el prefijo Ri, y los tipos con Rt. Por ejemplo si en RIB una función es Translate, en C es RiTranslate. Los argumentos en C deben ser enviados como se manejan en las funciones de éste es decir separados por comas encerrados en paréntesis a diferencia de RIB que no es necesario. Las funciones que reciben una lista de parámetros (por ejemplo la creación de primitivas) deben indicar la

finalización de la entrada de parámetros por medio del token RI_NULL, como se puede observar en el ejemplo, es decir si la línea de creación de una esfera en rib era:

```
Sphere 0.4 –0.4 0.4 360
```

En C sería:

```
RiSphere(0.4, -0.4, 0.4, 360.0, RI_NULL);
```

Para una documentación más amplía sobre el enlace con C consultar el al documento "The Renderman Interface" proveído por Pixar.

4.1.3 Estado de los Gráficos

La interfaz Renderman es similar a otros paquetes gráficos en mantener un estado de los gráficos. El estado de los gráficos contiene toda la información necesaria para generar una escena.

El estado de los gráficos esta divido en dos partes: un estado global el cual se mantiene constante mientras se genera una imagen y el estado actual que cambia de primitiva a primitiva generada. Los parámetros en el estado global son conocidos como opciones (options), mientras que los parámetros en el estado actual se conocen como atributos (attributes). Las opciones incluyen los parámetros de la cámara y de despliegue (display) y otros parámtros que afectan el tipo y calidad de generación de la escena en general (por ejemplo nivel global de detalle y número de colores de muestreo, etc). Los atributos

incluyen los parámetros que controlan la apariencia o sombreado (ejemplo la opacidad, color, material, luces, etc), y la matriz de modelado actual. Para ser adicionados en la especificación de modelos jerárquicos el estado de los gráficos puede ser cargado o sacado de la pila actual del estado de los gráficos.

El estado de los gráficos se maneja mediante el modo de interfaz es decir que permite ingresar o salir de un estado, esto mediante parejas Begin-End que especifican el inicio y salida de un nuevo estado.
A continuación se mostraran las diferentes parejas Begin-End que permiten entrar y salir de un estado de gráficos a otro.

RiBegin (RtToken name)
RiEnd (void)

RiBegin se utiliza para definir el inicio de un contexto de renderización, y para establecer todos los valores y atributos a su valor por defecto. Esta función se utiliza cuando se generan escenas utilizando el enlace con el lenguaje C debido a que en este se pueden tener diferentes contextos de renderización a diferencia del formato RIB, el cual permite especificar una escena o una secuencia de estas pero no un nuevo contexto, por lo que no existe equivalente en RIB para esta función.

El parámetro de entrada name que recibe la función toma significado según la librería de enlace utilizada, en el caso de BMRT este parámetro indica el comando encargado de la generación de la escena, es decir rendrib, aunque puede también utilizarse rgl, comando que genera la escena en opengl, o especificar el nombre del archivo de salida, el cual lo escribirá en formato rib.

Ejemplo en C:

```
RiBegin (“rgl”);
   ...
   ...
RiEnd();
```

Este ejemplo define que el comando encargado de la generación entre el par Begin-End será rgl, si se quisiera generar un archivo .rib con la información de la escena entre el par sería como sigue:

```
RiBegin (“prueba.rib”);
   ...
   ...
RiEnd();
```

Esto generaría un archivo prueba.rib que contendría la escena descrita en formato RIB.

Es conveniente recordar que el significado del parámetro depende de la librería de enlace utilizada, estos ejemplos sirven para BMRT y quizá para otros programas de generación pero de ninguna manera se debe generalizar.

RiEnd termina el contexto actual, el cual se encarga de limpiar y dejar los recursos utilizados libres.

Todas las funciones de Renderman deben ser llamados dentro de esta pareja Begin-End a excepción de RiErrorHandler, RiOption y RiContext.

RtContextHandle RiGetContext (Void)

Esta función retorna el contexto de renderización actual, si no hay uno activo retorna RI_NULL. (no existe en RIB)

Ejemplo en C:

```
RtContextHandle contexto_actual;

contexto_actual = RiGetContext();
```

```
RiContext (RtContextHandle handle)
```

Este comando recibe un RtContextHandle y lo activa como el actual sin destruir el existente. (no existe en RIB)
Ejemplo en C:

```
 RiContext (contexto_actual);
```

RiFrameBegin (RtInt frame)
RiFrameEnd ()

Esta pareja Begin-End define el comienzo y finalización de un frame o cuadro de una secuencia de animación, donde el parámetro de entrada especifica el número del cuadro.

El estado actual de los gráficos es almacenado cuando se llama RiFrameBegin y restaurado cuando se llama RiFrameEnd.

Esta función si está disponible en RIB y se muestra con un ejemplo:

```
FrameBegin 1
   ...
   ...
FrameEnd
```

RiWorldBegin ()
RiWorldEnd ()

Cuando es llamada RiWorldBegin todas las opciones del estado de gráficos son congeladas hasta que se finaliza la generación de la escena cuando se llama RiWorldEnd, las primitivas que se definan en la escena deben estar dentro de esta pareja de funciones, de lo contrario comenzarían a ser renderizadas inmediatamente, una manera de evitar este inconveniente es por medio de RiObjectBegin y RiObjectEnd, los cuales permiten definir un objeto sin generarlo retornando un handle o manejador a éste que puede ser llamado dentro de la pareja WorlBegin-WorldEnd para la generación.

Ejemplo en RIB:

```
WorldBegin
  ...
WorldEnd
```

4.1.3.1 Opciones (Options)

El estado de los gráficos tiene varias opciones las cuales deben ser inicializadas antes de renderizar un frame o cuadro. El grupo completo de opciones incluyen: una descripción de la cámara, una descripción del display o despliegue, al igual que controles de renderización en tiempo de ejecución como el algoritmo de caras ocultas a usar.

4.1.3.1.1 Cámara (camera)

El estado de los gráficos contiene un grupo de parámetros que definen las propiedades de la cámara, el modelo de cámara soporta planos de corte lejano y cercano (near, far) los cuales son perpendiculares a la dirección de la vista, así como un número arbitrario de planos de corte especificados por el usuario; la profundidad de campo es especificada por medio de un valor llamado f-stop, la distancia focal trabaja de manera similar que en una cámara real, es decir los objetos fuera de la distancia focal se verán borrosos, mientras que los objetos dentro de ella estarán enfocados; además la cámara permite manejo del tiempo de apertura del diafragma (shutter) para lograr efecto de desenfoque de movimiento (blur).

Renderman maneja un sistema de coordenadas basado en 6 marcos de referencia los cuales van desde el sistema de coordenadas de los objetos hasta el NDC (normalized device corrdinates). Estos sistemas de coordenadas son:

Objetos

Mundo

Cámara

Pantalla

Raster

NDC

Los comandos utilizados para el manejo de la cámara son los siguientes:

RiFormat(RtInt x, RtInt y, RtFloat pixelaspectratio)

Define el tamaño en pixeles de salida de la imagen a ser generada.

RiFrameAspectRatio(RtFloat frameaspectratio)

Define la proporción entre el ancho por el alto de la imagen.

RiScreenWindow(RtFloat left,RtFloat right,RtFloat bottom,RtFloat top)

Define un rectángulo en el plano de la imagen el cual es mapeado al sistema de coordenada raster. Este corresponde al área de despliegue seleccionada.

RiCropWindow(RtFloat xmin,RtFloat xmax,RtFloat ymin,RtFloat ymax)

Define un subrectángulo de la imagen ha ser renderizado. Generalmente se utiliza para visualizar un segmento de la imagen. Los valores son fracciones de la ventana de raster, es decir que si se desea generar solo la mitad superior los valores serían como xmin=0 , xmax=1, ymin= 0, ymax=0.5

RiProjection(RtToken name,...listadeparametros...)

La proyección determina cómo el sistema de coordenadas es convertido a las coordenadas de pantalla, usando el tipo de proyección y los planos cercano y lejano se crea una matriz de proyección la cual se concatena con la matriz actual de transformaciones. Los tipos de proyección que existen son perspectiva (perspective) y ortográfica (orthographic) y RI_NULL. Si se define una proyección perspectiva se puede definir además el ángulo del campo de vista (fov) en la lista de parámetros de la manera Token , Valor es decir por ejemplo "fov" , 45.

RiClipping(RtFloat near,RtFloat far)

Define los planos de corte cercano y lejano respectivamente.

RiClippingPlane(RtFloat x,RtFloat y,RtFloat z,RtFloat nx,RtFloat ny,RtFloat nz)

Define planos de corte especificados por el usuario, esto permite definir múltiples planos de corte. Donde x,y,z define un punto en la superficie del plano y nx,ny,nz el vector normal al plano.

RiDepthOfField(RtFloat fstop,RtFloat focalllength, RtFloat focaldistance)

La profundidad de campo define los objetos que se verán enfocados o borrosos dependiendo los parámetros de distancia focal y ampklitud del foco y f-stop.

RiShutter(RtFloat min,RtFloat max)

Esta función define el tiempo de apertura del diafragma de la cámara, Si min==max no se presentará desenfoque.

A continuación se presentarán una serie de ejemplos dónde se utilizan las funciones de manejo de cámara.

```
# Se define una ventana de 300x300 pixeles
Format 300 300 1

# La zona de despliegue va desde las coordenadas izq, der, abajo
# arriba

ScreenWindow -1 1 -1 1

# Define el numero de rayos por pixel
PixelSamples 3 3

WorldBegin

# Se mueve en el eje z en 1 unidad en el sistema de coordenadas de
la
# cámara

  Translate 0 0 1

  TransformBegin
    Color [0.95 0.95 0.95]
    Sphere 3000 -3000 3000 360
  TransformEnd

 Color [0 0 1]

 TransformBegin
    Translate 0 0 0
```

```
    Torus 0.4 0.15 00 360 360
  TransformEnd

WorldEnd
```

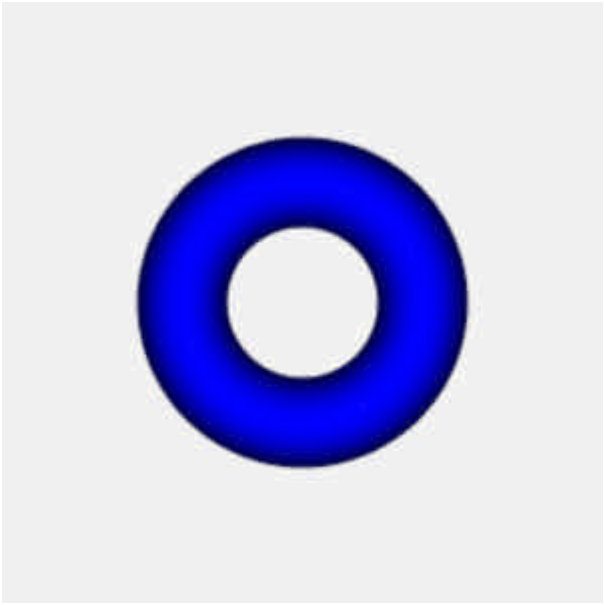


Figura 2: Ejemplo de Cámara Ortográfica.

Ahora veamos el mismo ejemplo pero modificando el FrameAspectRatio y ampliando el ScreenWindow, además de aplicar un CropWindow.

```
# Se define una ventana de 300x300 pixeles
Format 300 300 1

# La zona de despliegue va desde las coordenadas izq, der, abajo
# arriba

ScreenWindow -2 2 -2 2

# El AspectRatio es decir la relacion ancho/alto es de 0.5
# por eso la imagen se vera algo "achatada"

FrameAspectRatio 0.5
```

```
# la ventana de corte de rendering es la parte superior de
# la imagen, debe tenerse presenta que esta funcion trabaja
# con proporciones

CropWindow 0 1 0 0.5

# Posteriormente se explicara esta funcion
PixelSamples 3 3

WorldBegin

# Se mueve en el eje z en 1 unidad en el sistema de coordenadas de
la
# cámara

  Translate 0 0 1

  TransformBegin
    Color [0.95 0.95 0.95]
    Sphere 3000 -3000 3000 360
  TransformEnd

 Color [0 0 1]

 TransformBegin
    Translate 0 0 0
    Torus 0.4 0.15 00 360 360
  TransformEnd

WorldEnd
```

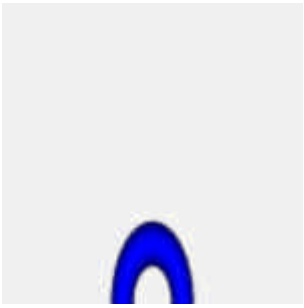


Figura 3: Ejemplo de Crop

Ahora veamos un ejemplo de proyección perspectiva

```
Format 300 300 1
ScreenWindow -1 1 -1 1
Projection "perspective"
PixelSamples 3 3

WorldBegin
  Rotate 5 1 0 0
  Translate -0.3 0 1

  TransformBegin
    Color [0.95 0.95 0.95]
    Sphere 3000 -3000 3000 360
  TransformEnd

 TransformBegin
    Color [0.7 0.7 1]
    Translate 0 -0.55 0
    Polygon "P" [10 0 10 -10 0 10 -10 0 -10 10 0 -10]
  TransformEnd

  TransformBegin
    Color [0.2 0 0.8]
    Translate 0 0 0
    Torus 0.4 0.15 0 360 360
  TransformEnd
```

```
  TransformBegin
    Color [0.2 0.6 0.1]
    Translate 1 0 2
    Rotate 90 1 0 0
    Cylinder 0.5 -0.55 0.55 360
  TransformEnd

  TransformBegin
    Color [1 0.2 0.1]
    Translate 2 -0.55 3
    Rotate -90 1 0 0
    Cone 1.1 0.5 360
  TransformEnd

WorldEnd
```

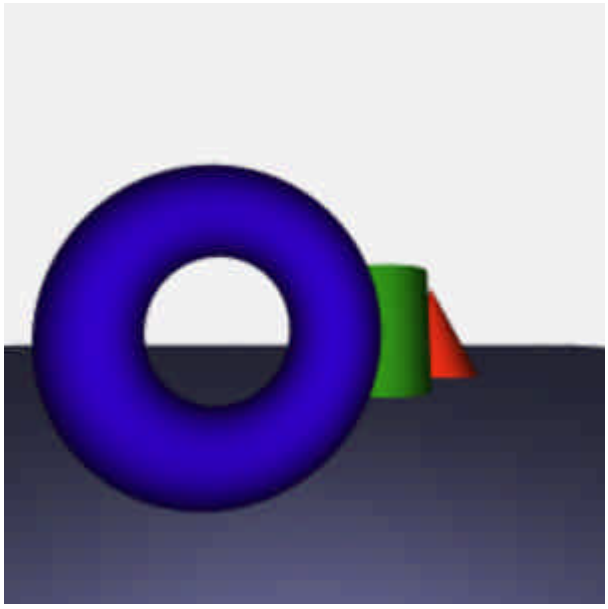


Figura 4: Ejemplo de Proyección Perspectiva

4.1.3.1.2 Despliegue (Display)

Los parámetros de despliegue en un programa de renderización son los encargados de definir como su nombre lo indica las opciones de despliegue

final de la imagen resultante, opciones tales como dispositivo de salida (como un archivo), canal alpha y filtros aplicados (como desenfoque gaussiano).

Los parámetros de despliegue generalmente no se utilizan en la generación normal de escenas, a excepción de la definición del dispositivo de salida, así que si se quiere ahondar en estas opciones se debe remitir a la especificación de Renderman.

La función encargada de definir el dispositivo de salida es:

RiDisplay (RtToken name, RtToken type, RtToken mode, ...paramlist...)

> Esta función recibe los parámetros name, type, mode y parámetros adicionales dependiendo del programa de renderización. El nombre define el nombre del archivo de salida o el nombre del framebuffer (buffer de generación), el tipo define el tipo de salida, o sea, archivo o ventana de despliegue, y el modo define el espacio de colores en el cual se generará la imagen (RGB, CMYK, N para escala de grises).

A continuación se muestra un ejemplo en el cual se genera una imagen a un archivo de imagen con extensión .tiff

```
Display "prueba.tif" "file" "rgb"

Format 300 300 1
ScreenWindow -1 1 -1 1
Projection "perspective"
```

```
PixelSamples 3 3

WorldBegin

  Rotate 5 1 0 0
  Translate 0 0 1

  TransformBegin
    Color [0.95 0.95 0.95]
    Sphere 3000 -3000 3000 360
  TransformEnd

  TransformBegin
    Color [0.3 0.3 1]
    Sphere 0.4 -0.4 0.4 360
  TransformEnd

 TransformBegin
    Translate 0 0.38 -0.3
    Color [1 0.3 0.4]
    Sphere 0.2 -0.2 0.2 360
  TransformEnd

 TransformBegin
    Color [0.7 0.7 1]
    Translate 0 -0.55 0
    Polygon "P" [10 0 10 -10 0 10 -10 0 -10 10 0 -10]
  TransformEnd

WorldEnd
```

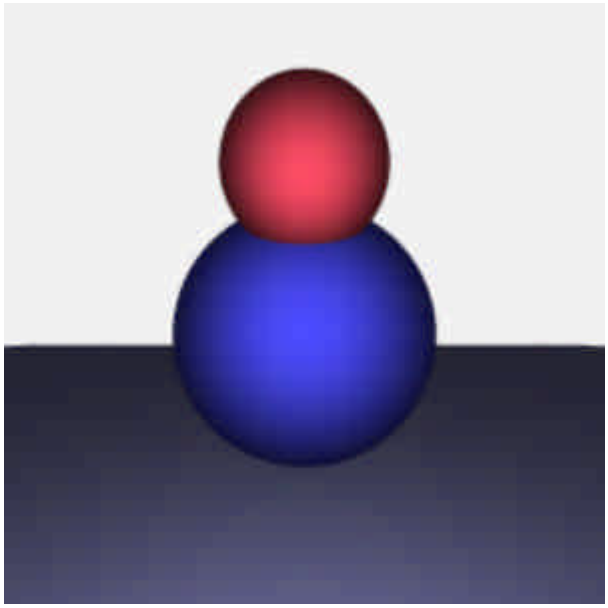


Figura 5: Ejemplo de Render almacenado en archivo Tiff

Este archivo generará una imagen llamada prueba.tif, si la generación de la imagen se está realizando con BMRT y el comando utilizado es "rendrib –d" la imagen no se grabará automáticamente, debe al finalzarse la generación presionar la tecla "w" para esto, o sino se omite la bandera –d y así la imagen no se generará en un framebuffer sino que se generará el archivo.

4.1.3.2. Atributos (Attributes)

Los atributos son parámetros en el estado de los gráficos que pueden cambiar mientras las primitivas geométricas están siendo definidas. Los atributos pueden ser grabados y restaurados con los siguientes comandos:

```
RiAttributeBegin ()
RiAttributeEnd ()
```

Estos comandos insertan y sacan los atributos y transformaciones de la pila de atributos y la de transformaciones respectivamente.

Los atributos son los encargados de la definición de colores, opacidad (nivel de transparencia), texturas, luces, definición de las características de las superficies, sombreados, entorno (atmósfera y neblina), brillos, nivel de detalle, grupos de objetos, orientación (mano-izquierda o mano-derecha) y lados de una superficie a ser renderizados.

La mejor manera de mostrar la utilización de estos atributos es por medio de ejemplos de escenas así que a continuación se dará un ejemplo comentado el cual ilustra algunos de los atributos que se pueden definir en una escena. (Los demás atributos definibles se pueden encontrar en la especificación de Renderman de Pixar).

El siguiente ejemplo muestra la manera de utilizar los atributos para definir propiedades como color, material, luces y mapas de desplazamiento, además de atributos propios del programa de renderización a utilizar.

```
##RenderMan RIB-Structure 1.0
version 3.03

##  Aquí se define el path a los materiales
Option "searchpath" "shader" [".:../shaders:../lib:&"]
Option "searchpath" "archive" [".:&"]
Format 600 350 1
PixelSamples 3 3
ShadingRate 1
Option "render" "max_raylevel" [2]

Display "vasegallery1.tif" "file" "rgba"
```

```
Projection "orthographic"
ScreenWindow -10 80 -40 10

ConcatTransform [1 0 0 0 0 -0.225304 -0.974288 0 0 0.974289 -0.225304 0 0 0
0 1]
Translate -0 -40 -15

WorldBegin

TransformBegin
CoordSysTransform "camera"

# Se define un atributo de fuente de luz:  Luz Ambiental
LightSource "ambientlight" 1 "intensity" [0.01]

# Se activa la luz numero 1 (anteriormente creada)
Illuminate 1 1

# Se define una luz secundaria enmarcada en un par Begin-End
# de atributos, es decir el estado de los graficos es congelado
# cuando entra a este par begin-end y restaurado al salir.

AttributeBegin
 Translate 800 1200 -1000
 LightSource "uberlight" 2 "string lighttype" ["omni"] "intensity" [0.8]
"lightcolor" [1 .95 .9]
AttributeEnd
Illuminate 2 1

# Aqui se define una tercera fuente de luz
AttributeBegin
 Translate -500 500 -1000
 Declare "lighttype" "string"
 LightSource "uberlight" 3 "lighttype" ["omni"] "intensity" [0.2] \
             "float nonspecular" [1] "lightcolor" [.95 .95 1]
AttributeEnd
Illuminate 3 1

# Se define una 4 luz
```

```
AttributeBegin
 Translate -600 600 900
 LightSource "uberlight" 4 "string lighttype" ["omni"] \
              "intensity" [0.35] "float nonspecular" [.7]
AttributeEnd
Illuminate 4 1

TransformEnd

##############################################################################

## Aqui comienza la definicion de las vasijas, como puede observarse
## se hace un llamado a un archivo externo en el cual esta definida
## la geometria de la vasija. (Se hizo esto para ahorrar espacio y
## para permitir una mejor visualizacion del ejemplo que se trata de
## definir atributos)

## Para cada vasija se define color y material
## con este ejemplo se pueden observar diferentes
## clases de material y como afectan sus coeficientes
## las propiedades del mismo.


AttributeBegin
 Translate 0 0 0
 Color 1 .3 .05
 Surface "roughmetal" "roughness" [0.3] "Ks" [1.5]
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 16 0 0
 Color 1 .3 .05
 Surface "shiny" "roughness" [0.3] "Ks" [1.5] "Kd" [0.15] "Kr" [0.4]
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 32 0 0
```

```
 Color 1 .3 .05
 Surface "plastic"
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 48 0 0
 Color 1 .3 .05
 Surface "shinyplastic"
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 64 0 0
 Color 1 .3 .05
 Surface "clay"
 ReadArchive "vase.rib"
AttributeEnd

##############################################################################


AttributeBegin
 Translate 0 0 -16
 TransformBegin
  Scale 3.5 3.5 3.5
  Surface "matte" "Kd" [0.8]
  # Aqui se aplica un mapa de desplazamiento al material
  Displacement "stucco" "float Km" [0.15]
 TransformEnd
 ReadArchive "vase.rib"
AttributeEnd


AttributeBegin
 Translate 16 0 -16
 TransformBegin
  Surface "matte" "Kd" [0.7]
  Displacement "castucco" "float Km" [0.04] "float freq" [1.5]
```

```
 TransformEnd
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 32 0 -16
 TransformBegin
  Scale 3.5 3.5 3.5
  Surface "plastic" "Kd" [0.7]

  # Aqui se define un atributo propio del programa de renderizacion
  # en este caso BMRT, aunque la gran mayoria utiliza los mismos
  # atributos

  Attribute "displacementbound" "coordinatesystem" ["shader"] "sphere" [.1]
  Displacement "dented" "float power" [1] "float Km" [0.1]
 TransformEnd
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 48 0 -16
 TransformBegin
  Scale 3.5 3.5 3.5
  Surface "plastic" "Kd" [0.7]
  Attribute "displacementbound" "coordinatesystem" ["shader"] "sphere" [.1]
  Displacement "dented"  "float power" [2] "float Km" [0.3]
 TransformEnd
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 64 0 -16
 TransformBegin
  Scale 3.5 3.5 3.5
  Surface "plastic" "Kd" [0.7]
  Attribute "displacementbound" "coordinatesystem" ["shader"] "sphere" [.1]
  Displacement "dented"  "float power" [3]
 TransformEnd
```

```
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 0 0 -32
 TransformBegin
  Scale 5 5 5
  Surface "veinedmarble"
 TransformEnd
 ReadArchive "vase.rib"
AttributeEnd


AttributeBegin
 Translate 16 0 -32
 TransformBegin
  Scale 3.5 3.5 3.5
  Surface "greenmarble"
 TransformEnd
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 32 0 -32
 TransformBegin
  Scale 10 10 10
  Surface "wood2"
 TransformEnd
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 48 0 -32
 TransformBegin
  Translate 7 7 0
  Scale 6 6 6
  Rotate 10 1 1 0
  Surface "oak" "float divotdepth" [0.007]
 TransformEnd
```

```
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Translate 64 0 -32
 TransformBegin
  Translate 10 10 0
  Scale 6 6 6
  Rotate 10 1 1 0
  Surface "oak" "float divotdepth" [0] "float roughness" [0.075] \
            "float Ks" [1.4]  "float Kd" [0.3] "float grainy" [0.5] \
            "color darkwood" [0.07 0.04 0.014]
 TransformEnd
 ReadArchive "vase.rib"
AttributeEnd

AttributeBegin
 Color .65 .65 .65
 Declare "visibility" "integer"
 Attribute "render" "visibility" [1]
 Surface "constant"
 Sphere 5000 -5000 5000 -360
AttributeEnd

WorldEnd
```

Para poder generar la imagen con el ejemplo anterior es necesario tener tanto los materiales como el archivo vase.rib, dichos archivos se encuentran en los ejemplos proveídos por BMRT.

La imagen generada por el ejemplo anterior se muestra en la figura 6.

Figura 6: Ejemplo de uso de materiales

4.1.4. Transformaciones

Las transformaciones son usadas para transformar puntos entre sistemas de coordenadas, el estado de los gráficos cuenta con la llamada "transformación actual" la cual es la matriz actual de transformación sobre la que se esté trabajando. Cada par Begin-End congela la matriz actual es decir la mete a la pila de transformaciones y la restaura al salir de él. Las funciones que se muestran a continuación concatenan matrices a la transformación actual.

`RiIdentity()`

Le asigna la identidad a la matriz actual.

`RiTransform (RtMatrix Trasnsform)`

Asigna la matriz transform a la matriz actual

RiTransform (RtMatrix Trasnsform)

Concatena la matriz transform a la matriz actual

RiPerspective (RtFloat fov)

Concatena la matriz de perspectiva fov a la transformación actual. El punto focal de perspectiva está en el origen y su dirección es a lo largo del eje-z. Para que esta función pueda ser llamada debe antes haberse llamado la función RiProjection "Perspective".

RiTranslate (Rtfloat dx, Rtfloat dy, Rtfloat dz)

Concatena una matriz de traslación sobre la matriz actual

RiRotate (Rtfloat angle, Rtfloat dx, Rtfloat dy, Rtfloat dz)

Concatena una matriz de rotación sobre la matriz actual.

RiScale (Rtfloat sx, Rtfloat sy, Rtfloat **sz)**

Concatena una matriz de escalado sobre la matriz actual.

RiSkew (Rtfloat angle, Rtfloat dx1, Rtfloat dy1, Rtfloat dz1,Rtfloat dx2, Rtfloat dy2, Rtfloat dz2)

Concatena una matriz de afilado sobre la matriz actual.

Las operaciones de transformación son quizá las que se manejan con mayor frecuencia por lo que a través de este documento se ha hecho uso de estas en los diferentes ejemplos dados.

Es importante notar que las operaciones de transformación actúan sobre el sistema de coordenadas que se esté manejando, es decir sobre la transformación actual por lo que al entrar y salir de un par begin-end dentro del cual se apliquen estas transformaciones se congela y restaura el estado de los gráficos para cada sistema de coordenadas.

4.1.5. Primitivas Geométricas

La interfaz Renderman soporta solamente definición de primitivas sólidas y superficies. Las primitivas sólidas son creadas a partir de superficies y conjunto de operaciones sobre ellas. Las Primitivas geométricas incluyen:

- Polígonos planos convexos, así como polígonos cóncavos con agujeros
- Poliedros (colecciones de polígonos planos con hoyos y vértices compartidos)
- Parches de maya y bilineares
- Parches bicúbicos y parches de maya con una base arbitraria
- Superficies de B-Spline racionales no uniformes de grados arbitrarios (NURBS)
- Superficies quadricas, toroides y discos
- Superficies de mayas de subdivisión

- Superficies implícitas
- Curvas de 1d o 2d o cintas

Todas las primitivas son creadas con la información de los vectores normales por lo que no es necesario definirlos, en caso de los polígonos formados por puntos, aquellos que hayan sido definidos en el sentido de las agujas del reloj tendrán un vector normal apuntando hacia la cámara por lo que sólo la cara que esté de ese lado podrá ser visualizada, en caso que en las opciones de despliegue haya sido habilitada la opción **`sides 2`** ambas caras del polígono podrán ser visualizadas.

Polígonos

`RiPolygon (RtInt nvertices, ...paramlist...)`

nvertices es el número de vértices en un polígono convexo simple cerrado.

No se realiza un chequeo por parte de la interfaz Renderman para asegurarse que se ha definido un polígono cerrado por lo que se podrían obtener resultados no esperados. El número de vértices en RIB no debe ser proveído, solamente basta con el token "P" y el valor que toma este token. Además de los vértices del polígono se puede definir las normales de cada vértice para el sombreado (Gouroud o

Phong) además de los niveles de opacidad, todo por medio de los tokens “N” “Cs” y “Os” respectivamente.

Ejemplo:

```
Format 300 300 1
ScreenWindow -3 3 -3 3

Projection "perspective"

WorldBegin

  Translate 0 0 2

  TransformBegin
    Color [1 1 1]
    Sphere 3000 -3000 3000 360
  TransformEnd

  TransformBegin
     Color [0 0 1]
     Rotate 90 1 0 0
     Translate 2.0 -0.55 0
     ## Ejemplo de poligono con sombreado phong
     Polygon "P" [2 0 2 -2 0 2 -2 0 -2 2 0 -2] "N" [0 0 -1 0 0 0 0 0 0 1 0
0]
  TransformEnd

  TransformBegin
     Rotate 90 1 0 0
     Translate -2.2 -0.55 0
     ## Ejemplo de poligono con sombreado gouraud
     Polygon "P" [2 0 2 -2 0 2 -2 0 -2 2 0 -2] "Cs" [2 0 0 0 2 0 0 0 2 0 2
2]
  TransformEnd

WorldEnd
```

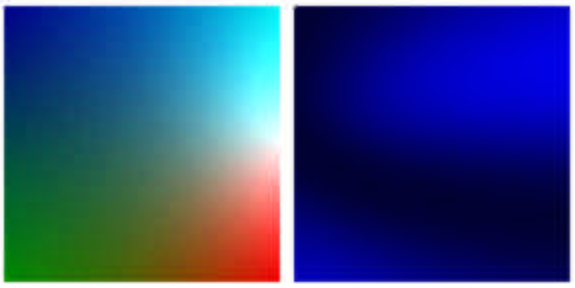


Figura 7: Ejemplo de creación de polígonos con diferente sombreado.

GeneralPolygon (RtInt nloops, RtInt nvertices[],.parameterlist.)

Defiene un polígono plano cóncavo con agujeros. Este polígono es especificado dando nloops listas de vértices (En RIB no es necesario especificarlas). El primer loop o ciclo es el límite exterior del polígono; todos los loops adicionales son agujeros. El vector nvertices contiene el número de vértices en cada loop, y tiene longitud nloops. Los vértices en cada ciclo son concatenados dentro de un simple vector de vértices.

Ejemplo:

```
Format 200 200 1
ScreenWindow -0.5 0.5 -0.5 0.5

PixelSamples 2 2

WorldBegin

 Translate -0.5 -0.5 1
```

```
Rotate 90 0 1 0

  TransformBegin
    Color [1 1 1]
    Sphere 3000 -3000 3000 360
  TransformEnd

  TransformBegin
     GeneralPolygon [4 3] "P" [0 0 0 0 1 0 0 1 1 0 0 1
                               0 0.25 0.5 0 0.75 0.75 0 0.75 0.25]

                              "Cs" [2 0 0 0 2 0 0 0 2 0 2 2
                                   2 0 0 0 2 0 2 0 2 ]
                              "N" [2 0 0 0 2 0 0 0 2 0 2 2
                                   2 0 0 0 2 0 2 0 2 ]
                              "Os" [2 0 0 0 2 0 0 0 2 0 2 2
                                   2 0 0 0 2 0 0.5 0 0.5 ]
  TransformEnd
```

4.1.3.2 WorldEnd

Figura 8: Ejemplo de polígono cóncavo con agujeros.

RiPointsPolygons (RtInt npolys, RtInt nvertices[], RtInt vertices[], ..parameterlist..)

Define npolígonos planos convexos que comparten vértices (en RIB no es necesario definirlo). El vector nvertices contiene el número de vertices en cada polígono y tiene longitud npolys. El vector vertices contiene un índice a cada vértice que forma cada polígono, los cuales serán definidos en la lista de parámetros que sigue después del vector.

Ejemplo

```
Format 200 200 1
ScreenWindow -2 2 -2 2

PixelSamples 2 2

WorldBegin

 Translate -0.5 -2 1
 Rotate 90 0 1 0

  TransformBegin
    Color [1 1 1]
    Sphere 3000 -3000 3000 360
  TransformEnd

  TransformBegin
     PointsPolygons [3 3 3] [0 3 2 0 1 3 1 4 3]
                    "P" [0 1 1 0 3 1 0 0 0 0 2 0 0 4 0]
                    "Cs" [0 0.3 0.4 0 0.3 0.9 0.2 0.2 0.2 0.5 0.2 0 0.9
0.8 0]
  TransformEnd
```

```
WorldEnd
```

Figura 9: Ejemplo de polígono convexo.

La figura 9 se compone de 3 polígonos que comparten los vértices 0, 1 y 3 entre sí, donde los vértices del lado derecho de arriba hacia abajo son el 0 y 1, y los del lado izquierdo son el 2, 3 y 4 igualmente de arriba hacia abajo. El primer polígono se forma con los vértices 0, 3, y 2; el segundo con los vértices 0, 1 y 3; y el tercero con los 1, 4 y 3. Como se puede observar en la lista de parámetros se definen los cuatro vértices y las propiedades de color del polígono.

`RiPointsGeneralPolygons (RtInt npolys, RtInt nloops, RtInt nvertices[], RtInt vertices[], ..parameterlist..)`

Esta función es similar a la anterior, sólo que trabaja con polígonos cóncavos con agujeros donde nloops indica el número

de ciclos comprendidos para cada polígono, para explicarlo más claro demos un ejemplo.

Ejemplo:

```
Format 200 200 1
ScreenWindow -1 1 -1 1

PixelSamples 2 2

WorldBegin

 Translate -0.5 -1 1
 Rotate 90 0 1 0

  TransformBegin
    Color [1 1 1]
    Sphere 3000 -3000 3000 360
  TransformEnd

  TransformBegin
     Color [1 0 0]
     PointsGeneralPolygons [2 2][4 3 4 3] [0 1 4 3 6 7 8 1 2 5 4 9
10 11]
                 "P" [ 0 0 1 0 1 1 0 2 1 0 0 0 0 1 0 0 2 0
                       0 0.25 0.5 0 0.75 0.75 0 0.75 0.25
                       0 1.25 0.5 0 1.75 0.75 0 1.75 0.25 ]
  TransformEnd

WorldEnd
```

Figura 10: Ejemplo de polígonos cóncavos con agujeros que comparten vértices.

El primer vector (nloops) indica cuantos loops hay para cada polígono, en este caso 2 loops para cada uno, es decir que del vector siguiente tomará el primer loop como el contorno y el segundo como agujeros, en este caso sería un polígono de 4 vértices como contorno y otro de 3 como agujero y así también para el segundo polígono; posteriormente se entrega en la lista de parámetros los vértices que componen los polígonos.

4.2 Blue Moon Renderign Tools (BMRT)

BMRT es una colección de programas de renderización los cuales implementan la interfaz estándar Renderman de Pixar.

El grupo de herramientas consiste de una completa implementación del estándar Renderman el cual soporta trazador de rayos, radiosidad, áreas de fuentes de luz, mapeo de texturas y entorno, sombreados programables en el lenguaje de sombreado de Renderman, desplazamientos, desenfoque de

movimiento, emisión de rayos de sombras automática, CSG, profundidad de campo, soporte de sombreado de imágenes y volúmenes, y otras características avanzadas de Renderman.

BMRT también contiene un rápido previsualizador de escenas que usa OpenGL tal que permite dar un vistazo de las escenas y las animaciones.

BMRT Funciona bajo las siguientes plataformas:

- SGI bajo IRIX 6 (en mips2, mips3, y mips4)
- Linux, corriendo bajo Intel y Alpha (FreeBSD, y NetBSD pueden correr con la versión para Intel Linux)
- Sun SPARC (corriendo SunOS o Solaris)
- DEC Alpha (están disponible las versiones para DEC OSF/1 y Linux/Alpha)
- Windows 95/98/NT (Intel)

4.3 OpenGL

OpenGL es una librería formada por 120 funciones aproximadamente. Entre las principales acciones que es posible realizar mediante comandos OpenGL se incluyen la especificación de los objetos y operaciones necesarias para producir aplicaciones interactivas en 3D.

OpenGL se ha diseñado de forma independiente del hardware de soporte con el fin de ser implementado sobre diferentes plataformas. Por este motivo, no

hay comandos para ejecutar tareas dependientes del hardware disponible como la gestión de ventanas o la lectura de datos de entrada del usuario.

OpenGL tampoco provee de comandos de alto nivel para describir modelos en 3D. El modelo debe construirse a partir de primitivas geométricas simples como puntos, líneas y polígonos.

4.3.1. OpenGL como una máquina de estados

OpenGL tiene un comportamiento similar al de una máquina de estados. OpenGL mantiene en todo momento un conjunto de variables que representan el estado actual. El valor de cada una de estas variables puede cambiarse mediante comandos de OpenGL.

El color, por ejemplo, es una variable de estado. Cuando se especifica un color todos los objetos son dibujados utilizando ese color mientras no se indique uno distinto. También hay variables de estado que hacen referencia a modos que pueden ser habilitados o deshabilitados con los comandos glEnable () o glDisable().

Todas las variables de estado tienen un valor actual que puede consultarse utilizando los comandos glGetBooleanv(), glGetFloatv() o glGetIntegerv(). Debe utilizarse el comando en función del tipo de datos de la variables de estado tienen un comando específico para obtener su valor, como por ejemplo: glGetLight*(), glGetError() o glGetPolygonStipple(). El estado puede salvarse y restaurarse en una pila de atributos utilizando los comandos glPushAttrib() y glPopAttrib().

4.3.2. Librerías

OpenGL dispone de un conjunto de comandos potente pero simple, que puede utilizarse para crear comandos de dibujo de más alto nivel. Se han desarrollado distintas librerías que aumentan la potencia de OpenGL:

- OpenGL Utility Library (GLU). Contiene rutinas que usan comandos OpenGL a bajo nivel para ejecutar tareas como inicializar matrices de especificación de vistas, proyecciones, etc.

- OpenGL Extensión del X Window System proporciona comandos para crear un contexto y asociarlo a una ventana de dibujo en una máquina que utilice X Window.

- The OpenGL Programming Guide Auxiliary Library fue desarrollada como soporte didáctico del libro "OpenGL Programming Guide", facilitando la implementación de ejemplos simples de programación en OpenGL.

- Open Inventor es una herramienta orientada a objetos basada en OpenGL que proporciona objetos y métodos para crear aplicaciones gráficas interactivas en 3D.

4.3.3. La guía auxiliar de programación de la librería OpenGL

Los comandos de OpenGL están diseñados para ser independientes de cualquier sistema de ventanas y de cualquier sistema operativo. En consecuencia, no hay comandos para abrir ventanas o capturar eventos del

teclado o el ratón. Para facilitar la realización de ejercicios de prácticas, ésta librería contiene este tipo de comandos, por lo que será dependiente del software utilizado. Junto a los comandos de control de ventanas y captura de eventos, la librería también incluye algunas rutinas para crear objetos 3D como esferas, cilindros, toros, etc. Los comandos incluidos en la librería son:

Control de Ventanas

- auxIntWindow() abre una ventana en la pantalla. Habilita la tecla Escape para salir de la aplicación y pone el color de fondo de la ventana a negro.

- AuxIntPosition() para especificar la posición y tamaño iniciales de la próxima ventana a crear.

- AuxIntDisplayModel() para especificar si se desea crear una ventana de ventana de color real (RGBA) o basada en paleta (color – index).

Capturando Eventos

- AuxReshapeFun() indica que la acción debe realizarse cuando la ventana se traslade por el escritorio, se cambie su tamaño o recupere el control y pase a primer plano.

- AuxKeyFunc() y auxMouseFunc() permiten asociar una rutina a una acción del teclado o del ratón.

4.3.4. Dibujando Objetos 3D

Los objetos que incluye la librería auxiliar son: esfera, cubo, toro, cilindro, cono, octaedro, dodecaedro, icosaedro y tetera. Se pueden dibujar tanto en modelo de alambres como en modelo sólido. Por ejemplo, para la esfera se dispone de los comandos:

- void auxWireSphere (Gldouble radius);
- void auxSolidSphere(Gldouble radius);

Todos los objetos se dibujan centrados en el origen de coordenadas.

4.3.5. Descripción de puntos, líneas y polígonos

- Punto

  Un punto queda definido por sus coordenadas en tres dimensiones. Cuando un punto se utiliza para definir líneas o polígonos, recibe el nombre de vértice.

- Líneas

  Llamadas segmentos, pues no son de longitud infinita. Una línea se define como la porción de recta comprendida entre dos vértices.

- Polígonos

Área cerrada formada por segmentos, teniendo en cuenta las siguientes limitaciones:

1. Los segmentos que forman el polígono no pueden cortarse entre sí.
2. El polígono resultante debe ser convexo.
3. No pueden describirse agujeros en el interior de un polígono.

Sin embargo, no hay restricción en cuanto al número de lados.

- Rectángulos

Para dibujar un rectángulo es preferible utilizar el comando glRect(), mucho más eficiente que el comando general para dibujar polígonos.

La sintaxis de este comando es:

```
void glRect{sifd}      (TYPEx1, TYPEy1, TYPEx2, TYPEy2);
void glRect{sifd}v     (TYPE*v1, TYPE*v2);
```

- Curvas

Se dibujan por aproximación utilizando segmentos pequeños. De igual forma, las superficies curvas se aproximan mediante polígonos pequeños(habitualmente triángulos o cuadriláteros)

4.3.5.1. ¿ Cómo especificar los vértices de un objeto ?

Todos los objetos de OpenGL se describen como un conjunto ordenado de vértices. Para indicar cada uno de ellos se utiliza el comando glVertex(), cuya sintaxis es la siguiente:

Void glVertex{234} {sfid} [v] (TYPEcoords);

Para indicar el principio de y el final del conjunto de vértices se utilizan los comandos glBegin() y glEnd().

4.3.5.2. Dibujo de Primitivas Geométricas

Ejemplo del polígono:

```
GlBegin (GL_POLYGON);
        glVertex2f (0.0, 0.0);
        glVertex2f (0.0, 3.0);
        glVertex2f (3.0, 3.0);
        glVertex2f (4.0, 1.5);
        glVertex2f(3.0, 0.0);
```

En la siguiente tabla se muestran los posibles valores que pueden pasarse como parámetro del comando glBegin() y el correspondiente tipo de primitiva que se generará como resultado:

Valor	Significado
GL_POINTS	Puntos individuales

GL_LINES	Cada par de vértices un segmento de línea
GL_POLYGON	Frontera de un polígono convexo
GL_TRIANGLES	Cada cuatro vértices un polígono de cuatro lados
GL_QUADS	Cada cuatro vértices un polígono de cuatro lados
GL_LINE_STRIP	Serie de segmentos conectados
GL_LINE_LOOP	Igual que la anterior pero une el último y el primero
GL_TRIANGLE_STRIP	Triángulos adyacentes
GL_TRIANGLE_FAN	Triángulos adyacentes con un vértice común
GL_QUAD_STRIP	Polígonos de cuatro lados adyacentes

4.3.5.3. Restricciones al Usar glBegin() y glEnd()

Entre un comando glBegibn () y su glEnd() correspondiente, además de las coordenadas de los vértices que definen el objeto, puede indicarse información adicional como el color del vértice, su normal asociada, o su relación con las coordenadas de textura. Los comandos utilizados para proporcionar esta información son los siguientes:

Comando	**Propósito**
GlVertex *()	Coordenadas
GlColor*()	Indice de color

GlNormal*()	Vector normal
GlEvalCoord*()	Genera coordenadas
GlCallList()	Ejecuta las display list
GlTextCoord*()	Coordenadas de textura
glEdgeFlag*()	Control dibujo de aristas
GlMaterial*()	material

No se puede incluir ningún otro tipo de comando entre glBegin() y glEnd(). Sin embargo, si se puede incluir código de programación. La restricción sólo afecta a comandos OpenGL.

4.3.5.4. Dibujo de puntos, líneas y polígonos

Por defecto, un punto se dibuja como un pixel, una línea se dibuja de forma sólida y de un pixel de ancho, y los polígonos se dibujan rellenos. A continuación se muestra como cambiar estas características por defecto.

- Sobre los Puntos

Para controlar el tamaño de un punto se utiliza el comando glPointSize(), con un único argumento que índica el ancho en pixels. Su sintaxis es la siguiente:

void glPointSize (Glfloat size);

- Sobre las líneas

Se puede especificar tanto el ancho de la línea como el patrón de dibujo (puntos, guiones o combinaciones de ambos).

Se utiliza el comando glLineWidth() con un único parámetro similar al analizado para los puntos. Su sintaxis es la siguiente:

void glLineWidth (Glfloat size);

Se utiliza el comando glLineStipple() para definir la trama, y glEnable() para habilitarla. Por ejemplo:

```
GlLineStipple (1, ox3f07);
GlEnable  (GL_LINE_STIPPLE);
```

El patrón es una serie de 16 ceros y unos. Un 1 indica que ha de dibujar, mientras que el cero indica que no, comenzando por el bit de mas bajo nivel.

El patrón puede ser escalado utilizando un factor, el cual multiplica cada subserie de unos y ceros. El factor de escala ha de ser un valor entre 1 y 255. La sintaxis es la siguiente:

Void glLineStipple (Glint factor, Glushaort pattern);

- Sobre los polígonos

Por defecto los polígonos se dibujan con borde y rellenos del color de dibujo, pero no tiene porque ser así.

Un polígono tiene dos caras y puede dibujarse de forma distinta dependiendo de la cara que se vea. Por defecto ambas caras se dibujan da la misma forma. Para cambiar este comportamiento por defecto, así como para dibujar únicamente la frontera o los vértices del polígono, se utilizará el comando glPolygonMode(), cuya sintaxis es:

Void glPolygonMode (Glenum face, Glenum mode);

El parámetro **face** puede tener los siguientes valores: GL_FRONT, GL_BACK o GL_FRONT_BACK, indicando a cual de las dos caras del polígono, afectará la orden; el parámetro **mode** puede tomar los valores GL_POINT, GL_LINE o GL_FILL para dibujar los vértices, la frontera o el interior respectivamente.

Por norma, la cara del polígono cuyos vértices aparecen en orden contrario a las agujas del reloj es la cara frontal. Se puede construir una superficie a base de polígonos si utilizamos una orientación consistente, es decir, se definen los vértices siempre en el sentido de las agujas del reloj o siempre al contrario.

Para especificar que cara es la frontal se ha de utilizar la función glFrontFace() e indicar como parámetro el sentido: GL_CW (ClockWise o horario) o GL_CCW (CounterClockWise o antiahorro). Su sintaxis es:
Void glFrontFace (Glnum mode);

Si una superficie construida a base de polígonos está totalmente cerrada, nunca podremos ver aquellos situados a espaldas del observador, es decir a

OpenGL que elimine los polígonos a espaldas para aumentar la velocidad de dibujo.

Con el comando glCullFace() se indica qué polígonos se han de eliminar, y con glEnable() se habilita esta opción del OpenGL. La sintaxis es al siguiente:

Void glCullFace (Glenum mode);

El parámetro **mode** puede tomar los valores GL_FRONT, GL_BACK o GL_FRONT_AND_BACK. Como parámetro de glEnable(), se ha de pasar GL_CULL_FACE.

Los polígonos pueden rellenarse con un patrón de 32x32 bits, el cual se especifica mediante la función glPolygonStipple(), cuya sintaxis es:
Void glPolygonStipple (const Glubyte *mask);

El parámetro **mask** es un puntero a un bitmap de 32x32, que se crea de forma similar a la analizada para los patrones de dibujo de líneas. Para activar el rellenado por patrón es necesario activarlo mediante el comando glEnable() con el parámetro GL_POLYGON_STIPPLE.

Para activar el rellenado con el patrón definido utilizaremos los comandos:

GlPplygonStipple (mosca);
GlEnable (GL_POLYGON_STIPPLE);

4.3.5.5. Vectores Normales

Un vector normal, o simplemente una normal, es un vector que señala una dirección y que es perpendicular a una superficie. Para una superficie plana, con una normal es suficiente para cualquier punto de ella, pero para una superficie curva, la normal depende de cada uno de los vértices de un polígono. O bien algunos o todos ellos pueden compartir una normal, pero no puede especificarse una normal en ningún otro sitio, es decir, siempre debe darse la norma a un vértice entre una pareja de comandos glBegin() y glEnd().

Las normales se utilizan para determinar que cantidad de luz recibe el objeto en un punto. Se utiliza el comando glNormal*() para especificar una normal, y depués se indican los vértices que tiene dicha normal. Su sintaxis es:

Void glNormal3 (BSIDF) (TYPEnx, TYPEny, TYPenz);
Void glNormal3 (bsidf) v (const TYPE*v);

4.4. LENGUAJE UNIFICADO DE MODELADO "UML".

"UML" son las siglas de Unifed Modeling Language.

Es un lenguaje para especificar, construir, visualizar y documentar los artefactos de un sistema de software orientado a objetos (OO). Un artefacto es un modelo que representa una información que es utilizada o producida mediante un proceso de desarrollo de software.

UML no define una metodología estándar que determine las fases de desarrollo de un sistema, sino únicamente un lenguaje de modelado, El UML recomienda utilizar los procesos que otras metodologías tienen definidos.

Las empresas pueden utilizar UML como el lenguaje para definir sus propios procesos y lo único que tendrán en común con otras organizaciones que utilicen UML serán los tipos de diagramas. UML es un método independiente del proceso. Los procesos de desarrollo deben ser definidos dentro del contexto donde se van a implementar los sistemas.

4.4.1. Inicio de UML

A partir del año 1994, Grady Booch y Jim Rumbaugh (creador de OMT) se unen en una empresa común, Rational Software Corporation, y comienzan a unificar sus dos métodos. Un año más tarde, en octubre de 1995, aparece UML (**Unified Modeling Language**) 0.8, la que se considera como la primera versión del UML. A finales de ese mismo año, Ivar Jacobson, creador de OOSE (**Object Oriented Software Engeneering**) se añade al grupo, para posteriormente dar inicio a lo que hoy se conoce como UML.

4.4.2. Objetivo de UML

El objetivo central del lenguaje es abstraer cualquier tipo de sistema, sea informático o no, mediante diagramas. Un diagrama es una representación gráfica de una colección de elementos del modelo, que habitualmente toma

forma de grafo donde los arcos que conectan sus vértices son las relaciones entre los objetos y los vértices se corresponden con otros elementos del modelo.

4.4.3. Modelado de objetos

En la especificación de UML podemos comprobar que una de las partes que lo componen es un metamodelo formal. Un metamodel **o** es un modelo que define el lenguaje para expresar otros modelos.
Una parte de UML define una abstracción con significado de un lenguaje para expresar otros modelos (es decir, otras abstracciones de un sistema, o conjunto de unidades conectadas que se organizan para conseguir un propósito); en otras palabras, UML define un lenguaje con el que podemos abstraer cualquier tipo de modelo.

Con la creación de UML se persigue obtener un lenguaje que sea capaz de abstraer cualquier tipo de sistema, sea informático o no, mediante los diagramas, es decir, mediante representaciones gráficas que contienen toda la información relevante del sistema.

4.4.4. Artefactos para el Desarrollo de Proyectos

Los artefactos de UML se especifican en forma de diagramas, éstos, junto con la documentación sobre el sistema constituyen los artefactos principales que el modelador puede observar.

Entre los artefactos que usa UML podemos encontrar:

4.4.4.1 Diagramas de Casos de Uso

Un caso de uso es una secuenci a de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema . Es un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo.

Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar como reacciona una respuesta a eventos que se producen en él mismo. En este tipo de diagrama intervienen algunos conceptos nuevos: un **actor** es una entidad externa al sistema que se modela y que puede interactuar con él; por ejemplo un usuario o cualquier otro sistema. Las relaciones entre casos de uso y actores pueden ser las siguientes:

Un actor se comunica con un caso de uso.

Un caso de uso extiende otro caso de uso

Un caso de uso utiliza otro caso de uso.

4.4.4.2. Diagramas de Clases

Los diagramas de clases representan un conjunto de elementos del modelo que son estáticos: las clases y los tipos, sus contenidos y las relaciones que se establecen entre ellos.

Algunos de los elementos que se pueden clasificar como estáticos son los siguientes:

Paquete: Es el mecanismo de que dispone UML para organizar sus elementos en grupos, se representa un grupo de elementos del modelo. Un sistema es un único paquete que contiene el resto del sistema, por lo tanto, un paquete debe poder anidarse, permitiéndose que un paquete contenga otro paquete.

Clases: Una clase representa un conjunto de objetos que tienen una estructura, un comportamiento y unas relaciones con propiedades parecidas. Describe un conjunto de objetos que comparte los mismos atributos, operaciones, métodos, relaciones y significado. En UML una clase es una implementación de un tipo. Los componentes de una clase son:

Atributo. Se corresponde con las propiedades de una clase o un tipo. Se identifica mediante un nombre. Existen atributos simples y complejos.

Métodos. También conocido como operaciones, es un servicio proporcionado por la clase que puede ser solicitado por otras clases y que produce un comportamiento en ellas cuando se realiza.

Las clases pueden tener varios parámetros formales, son las clases denominadas plantillas. Sus atributos y operaciones vendrán definidas según sus parámetros formales. Las plantillas pueden tener especificados los valores

reales para los parámetros formales, entonces reciben el nombre de clase parametrizada instanciada . Se puede usar en cualquier lugar en el que se podría aparecer su plantilla.
Relacionado con las clases nos encontramos con el término utilidad, que se corresponde con una agrupación de variables y procedimientos globales en forma de declaración de clase, también puede definirse como un estereotipo (o nueva clase generada a partir de otra ya existente) de un tipo que agrupa variables globales y procedimientos en una declaración de clase. Los atributos y operaciones que se agrupan en una utilidad se convierten en variables y operaciones globales. Una utilidad no es fundamental para el modelado, pero puede ser conveniente durante la programación.

Metaclase: Es una clase cuyas instancias son clases. Sirven como depósito para mantener las variables de clase y proporcionan operaciones (método de clase) para inicializar estas variables. Se utilizan para construir metamodelos (modelos que se utilizan para definir otros modelos).
Tipos: Es un descriptor de objetos que tiene un estado abstracto y especificaciones de operaciones pero no su implementación. Un tipo establece una especificación de comportamiento para las clases.

Interfaz: Representa el uso de un tipo para describir el comportamiento visible externamente de cualquier elemento del modelo.

Relación entre clases: Las clases se relacionan entre sí de distintas formas, que marcan los tipos de relaciones existentes:

Asociación:

Es una relación que describe un conjunto de vínculos entre clases. Pueden ser binarias o n-arias, según se implican a dos clases o más. Las relaciones de asociación vienen identificadas por los roles , que son los nombres que indican el comportamiento que tienen los tipos o las clases, en el caso del rol de asociación (existen otros tipos de roles según la relación a la que identifiquen). Indican la información más importante de las asociaciones. Es posible indicar el número de instancias de una clase que participan en una relación mediante la llamada **multiplicidad**. Cuando la multiplicidad de un rol es mayor que 1, el conjunto de elementos que se relacionan puede estar ordenado. Las relaciones de asociación permiten especificar qué objetos van a estar asociados con otro objeto mediante un calificador. El calificador es un atributo o conjunto de atributos de una asociación que determina los valores que indican cuales son los valores que se asociarán.

Una asociación se dirige desde una clase a otra (o un objeto a otro), el concepto de navegabilidad se refiere al sentido en el que se recorre la asociación.

Existe una forma especial de asociación, la agregación, que especifica una relación entre las clases donde el llamado "agregado" indica el todo y el "componente" es una parte del mismo.

Composición:

Es un tipo de agregación donde la relación de posesión es tan fuerte como para marcar otro tipo de relación. Las clases en UML tienen un tiempo de vida determinado, en las relaciones de composición, el tiempo de vida de la clase que es parte del todo (o agregado) viene determinado por el tiempo de vida de la clase que representa el todo, por tanto es equivalente a un atributo, aunque no lo es porque es una clase y puede funcionar como tal en otros casos.

Generalización:

Cuando se establece una relación de este tipo entre dos clases, una es una superclase y la otra es una subclase. La subclase comparte la estructura y el comportamiento de la superclase. Puede haber más de una clase que se comporte como subclase.

Dependencia:

Una relación de dependencia se establece entre clases (u objetos) cuando un cambio en el elemento independiente del modelo puede requerir un cambio en el elemento dependiente.

Relación de Refinamiento:

Es una relación entre dos elementos donde uno de ellos especifica de forma completa al otro que ya ha sido especificado con cierto detalle.

- Diagramas de Interacción o Comportamiento

Los diagramas de interacción indican el flujo de mensajes entre elementos del modelo, el flujo de mensajes representa el envío de un mensaje desde un objeto a otro si entre ellos existe un enlace. Hay varios tipos y son:

4.4.4.3. Diagramas de secuencia

Muestran las interacciones entre un conjunto de objetos, ordenadas según el tiempo en que tienen lugar. En los diagramas de este tipo intervienen objetos, que tienen un significado parecido al de los objetos representados en los diagramas de colaboración, es decir son instancias concretas de una clase que participa en la interacción. El objeto puede existir sólo durante la ejecución de la interacción, se puede crear o puede ser destruido durante la ejecución de la interacción. Un diagrama de secuencia representa una forma de indicar el período durante el que un objeto está desarrollando una acción directamente o a través de un procedimiento.
En este tipo de diagramas también intervienen los mensajes, que son la forma en que se comunican los objetos: el objeto origen solicita (llama a) una operación del objeto destino. Existen distintos tipos de mensajes según cómo se producen en el tiempo: simples, síncronos, y asíncronos.

Los diagrama de secuencia permiten indicar cuál es el momento en el que se envía o se completa un mensaje mediante el tiempo de transición , que se especifica en el diagrama.

4.4.4.4. Diagramas de colaboración

Muestra la interacción entre varios objetos y los enlaces que existen entre ellos. Representa las interacciones entre objetos organizadas alrededor de los objetos y sus vinculaciones. A diferencia de un diagrama de secuencias, un diagrama de colaboraciones muestra las relaciones entre los objetos, no la secuencia en el tiempo en que se producen los mensajes. Los diagramas de secuencias y los diagramas de colaboraciones expresan información similar, pero en una forma diferente.

Formando parte de los diagramas de colaboración nos encontramos con objetos, enlaces y mensajes. Un objeto es una instancia de una clase que participa como una interacción, existen objetos simples y complejos. Un objeto es activo si posee un thread o hilo de control y es capaz de iniciar la actividad de control, mientras que un objeto es pasivo si mantiene datos pero no inicia la actividad.

Un enlace es una instancia de una asociación que conecta dos objetos de un diagrama de colaboración. El enlace puede ser reflexivo si conecta a un elemento consigo mismo. La existencia de un enlace entre dos objetos indica que puede existir un intercambio de mensajes entre los objetos conectados.
Los mensajes que se envían entre objetos pueden ser de distintos tipos, también según como se producen en el tiempo; existen mensajes simples, sincrónicos, balking , timeout y asíncronos.

4.4.4.5. Diagramas de actividad

Son similares a los diagramas de flujo de otras metodologías OO. En realidad se corresponden con un caso especial de los diagramas de estado donde los estados son estados de acción (estados con una acción interna y una o más transiciones que suceden al finalizar esta acción, o lo que es lo mismo, un paso en la ejecución de lo que será un procedimiento) y las transiciones vienen provocadas por la finalización de las acciones que tienen lugar en los estados de origen. Siempre van unidos a una clase o a la implementación de un caso de uso o de un método (que tiene el mismo significado que en cualquier otra metodología OO).

Los diagramas de actividad se utilizan para mostrar el flujo de operaciones que se desencadenan en un procedimiento interno del sistema.

4.4.4.6. Diagramas de estado

Representan la secuencia de estados por los que un objeto o una interacción entre objetos pasa durante su tiempo de vida en respuesta a estímulos (eventos) recibidos. Representa lo que podemos denominar en conjunto una máquina de estados. Un **estado** en UML es cuando un objeto o una interacción satisface una condición, desarrolla alguna acción o se encuentra esperando un evento.

Cuando un objeto o una interacción pasa de un estado a otro por la ocurrencia de un evento se dice que ha sufrido una **transición**, existen varios tipos de transiciones entre objetos: simples (normales y reflexivas) y

complejas. Además una transición puede ser interna si el estado del que parte el objeto o interacción es el mismo que al que llega, no se provoca un cambio de estado y se representan dentro del estado, no de la transición.

4.4.4.7. Diagramas de Implementación

Se derivan de los diagramas de proceso y módulos de la metodología de Booch, aunque presentan algunas modificaciones. Los diagramas de implementación muestran los aspectos físicos del sistema. Incluyen la estructura del código fuente y la implementación, en tiempo de implementación. Existen dos tipos:

4.4.4.7.1. Diagramas de componentes

Muestra la dependencia entre los distintos componentes de software, incluyendo componentes de código fuente, binario y ejecutable. Un componente es un fragmento de código software (una fuente, binario o ejecutable) que se utiliza para mostrar dependencias en tiempo de compilación.

4.4.4.7.2. Diagrama de plataformas o despliegue

Muestra la configuración de los componentes hardware, los procesos, los elementos de procesamiento en tiempo de ejecución y los objetos que existen en tiempo de ejecución. En este tipo de diagramas intervienen nodos, asociaciones de comunicación, componentes dentro de los nodos y objetos que se encuentran a su vez dentro de los componentes. Un **nodo** es un

objeto físico en tiempo de ejecución, es decir una máquina que se compone habitualmente de, por lo menos, memoria y capacidad de procesamiento, a su vez puede estar formado por otros componentes.

4.5. INGENIERÍA DEL SOFTWARE

Por medio de esta disciplina tecnológica y administrativa se lleva a cabo la evaluación y mejoramiento de los procesos de construcción del software.

Todo producto de software, es decir, el conjunto de programas fuentes y ejecutables, procedimientos, reglas, documentación asociada debe ser de buena calidad sistemática, en otras palabras, debe cumplir los estándares de calidad de la norma para desarrollo de software ISO 9001.

La norma para desarrollo de software, que define estándares de calidad de productos y procesos es la ISO 9001, la cual es una aplicación de la norma ISO 900-1.

También se debe tener en cuenta en el desarrollo de un software los modelos del ciclo de vida de éste, que comprende el proceso de desarrollo de software desde su nacimiento hasta su reemplazo o eliminación.

4.5.1. Paradigmas del Ciclo de Vida de Desarrollo de Software

- **Paradigma de Codificar y Corregir**

Cuando el programador codifica primero y deja análisis, especificaciones y pruebas para después.

Este enfoque presenta problemas como la falta de planificación, falsa sensación de productividad, después de varias correcciones el programa se queda sin estructuras y no se produce documentación.

- **Paradigma de cascada**

En este paradigma se reconocen las etapas para el desarrollo del software:

Análisis de requerimientos
Especificación de los requerimientos
Diseño externo o de la interfaz con el usuario y el diseño interno
Implementación (Codificación y pruebas)
Mantenimiento

La base de este paradigma es desarrollar cada etapa hasta el fin, en caso de pasar a otra etapa y encontrar errores en la anterior, hay que devolverse y corregirlos.

Presenta desventajas como el énfasis en la producción de documentos completamente elaborados, la no aplicabilidad a productos de software

altamente interactivos, dificultad de tener los requerimientos bien definidos al principio.

- **Paradigma de prototipo**

Es una mejora del paradigma de cascada. Puede tomar alguna de las siguientes formas:

Un escenario (simulación del uso del sistema)

Una demostración (porciones de código que realizan algunas funciones)

Una versión (aplicación liberada que puede usarse bajo condiciones preliminares añadiendo o quitando funciones existentes y creándole su documentación)

Uso

Se usa cuando los requerimientos no son claros o no se identifican en forma detallada los requerimientos de entrada, salida y funciones.

Las etapas de desarrollo de software con este paradigma son:

Recolección de requerimientos
Diseño rápido
Construcción de prototipo
Evaluación de requerimientos
Producto construído

Aquí también cuando se detecta un error en alguna etapa, se debe devolver a corregirlo, pero no necesariamente a la etapa anterior sino a cualquier etapa que ya se haya desarrollado.

Las desventajas de este enfoque son:

Crea expectativas mas allá de lo que realmente se puede hacer
Se dificulta la dirección y control del proceso de desarrollo más que en el método clásico

- **Paradigma case**

Este paradigma dedica la mayor parte del tiempo y esfuerzo al análisis y diseño y menos a la codificación y prueba. Se compone de las fases:

Generación de prototipos
Especificación del diseño
Verificación del diseño
Generación del código
Prueba del sistema
Sistema completo

Es importante anotar que esta metodología permite que las especificaciones puedan crearse interactivamente en forma de diagramas estructurados que representan estructuras de datos del programa, entidades de datos y funciones procedimentales.

- **Paradigma de Espiral**

Es el enfoque más realista para el desarrollo de software y sistemas a gran escala, ya que es una combinación de los modelos de cascada y prototipos con sus mejores características y además considera una etapa de análisis de riesgos. Consta de 4 actividades principales que se representan en 4 cuadrantes, los cuales se recorren incrementalmente en la espiral así:

Planeación
Análisis de riesgos
Ingeniería
Evaluación del cliente

4.5.2. Paradigmas para desarrollo de software

Los paradigmas de desarrollo de software utilizan diferentes metodologías (colección de métodos aplicados a lo largo del ciclo de vida de desarrollo del software) coherentes entre sí y siguen una filosofía o enfoque de desarrollo de software inicial. Los 3 enfoques existentes son:

- **Paradigma orientado por procedimientos o estructurado**

Este paradigma considera los procesos o funciones como la parte fundamental del modelo del sistema y los datos después. Las metodologías para análisis y diseño estructurado permiten la disgregación funcional para

construir modelos lógicos orientados según propósitos y requerimientos del sistema. Entre las metodologías utilizadas están el análisis estructurado de Gane Sarson y diseño estructurado de Yourdon.

- **Paradigma Orientado por Datos**

La parte más importante para este modelo son las entradas y las salidas, las estructuras y los datos se definen primero y los componentes procedimentales se derivan de la estructura de datos. Las metodologías que soportan este enfoque son el diseño estructurado de Jackson y las metodologías Warnier y Orr

- **Paradigma orientado por Objetos**

Considera la estructura de los objetos como parte fundamental del sistema. Las metodologías empleadas por este paradigma son la OMT y Booch.

5. Análisis del Sistema

5.1. Definición de los Requerimientos

5.1.1. Requerimientos Funcionales

1. El sistema debe estar constituido por los siguientes módulos:

- Módulo de creación de primitivas.
- Módulo de Manejo de las características del mundo (Luces y Cámara).
- Módulo de transformación de primitivas
- Módulo de visualización de escenas.
- Módulo de animación de escenas.
- Módulo de Renderización.
- Módulo de almacenamiento de escenas.

El Módulo de creación de primitivas debe estar en capacidad de:

2. Permitir la creación de primitivas caja con los siguientes parámetros: Nombre, color, alto, ancho, profundidad, x, y, z.

3. Permitir la creación de primitivas esfera con los siguientes parámetros: Nombre, color, radio, zmax, zmin, ángulo theta.

4. Permitir la creación de primitivas cono con los siguientes parámetros: Nombre, color, radio, altura, ángulo theta.

5. Permitir la creación de primitivas cilindro con los siguientes parámetros: Nombre, color, radio, zmax, zmin, ángulo theta.

6. Permitir la creación de primitivas disco con los siguientes parámetros: Nombre, color, radio, altura , ángulo theta.

7. Permitir la creación de primitivas toroide con los siguientes parámetros: Nombre, color, radio mayor, radio menor, ángulo theta, ángulo phi máximo y mínimo.

8. Permitir la creación de primitivas hiperboloide con los siguientes parámetros:
Nombre, color, posición x, y z de los puntos guía (puntos 1 y 2) y el ángulo entre ellos.

El Módulo de Manejo de las características del mundo (Luces y Cámara) debe estar en capacidad de:

9. Permitir la creación de una luz ambiental con parámetros de intensidad y color.

10. Permitir la creación de luces omnidireccionales con parámetros de intensidad, color, y posición en el espacio x, y, z.

11. Permitir la creación de luces direccionales tipo reflector (spot) con parámetros de intensidad, color, posición origen y posición destino, ángulo interno y externo del cono de luz y factor de decaimiento.

12. Definir la cámara en función de su posición en el espacio y hacia dónde se encuentra mirando.

El Módulo de transformación de primitivas debe estar en capacidad de:

13. Permitir la selección por nombre de uno o varios objetos para la posterior aplicación de alguna transformación o de un material.

14. Permitir la traslación en el espacio definiendo las coordenadas x, y, z del grupo de objetos seleccionados.

15. Permitir la rotación en un ángulo theta sobre un eje axial del grupo de objetos seleccionados.

16. Permitir el escalado del grupo de objetos seleccionados por un factor de escalación sobre los ejes x, y o z.

17. Permitir el cortado (sharing) de los objetos seleccionados dando como valores de entrada el eje sobre el cual se trabajará y el recorte para los otros dos ejes.

18. Permitir operaciones de reflexión de los objetos respecto a los ejes axiales.

19. Permitir la aplicación de un material de la librería proveída por el programa al grupo de objetos seleccionados, material al cual se le podrán realizar variaciones en sus coeficientes de luz ambiental, difusa y especular, así como el color y el color especular si el material así lo permite.

El Módulo de visualización debe estar en capacidad de:

20. Permitir visualizar los elementos gráficos de la escena por medio de cuatro paneles de OpenGL, tres de los cuales presentarán una vista en proyección ortográfica y el último mostrará el despliegue gráfico de la cámara la cual se encuentra en proyección de perspectiva.

21. Permitir la definición de los límites de visualización para cada panel de OpenGL, los cuales están expresados en valores de distancia cercana y lejana (near, far) además de permitir definir el valor de fov para el pánel correspondiente a la cámara, el color del fondo del panel y el tipo de despliegue, plano o en maya (flat o wire).

22. Permitir el despliegue en proyección ortográfica de los 6 volúmenes de vista básicos (arriba, abajo, izquierda, derecha, adelante y atrás) en los paneles de OpenGL encargados para esto.

23. Permitir la definición de un valor de acercamiento (zoom) a los volúmenes de vista y cámara.

24. Permitir el despliegue gráfico de la escena según el número de cuadro actual (Frame).

El Módulo de animación de escenas debe estar en capacidad de:

25. Permitir el desplazamiento a través de los diferentes cuadros que conforman la animación por medio de una interfaz que provea la funcionalidad de atrasar, adelantar, parar y avanzar.

26. Permitir la definición del número de cuadros que conforman la escena, además de definir la velocidad de despliegue cuadro a cuadro del módulo de visualización.

27. Permitir la creación de cuadros clave (KeyFrame) por medio de un botón de grabado, los cuales definirán dónde realmente han ocurrido cambios en los objetos o la cámara.

El Módulo de Renderización debe estar en capacidad de:

28. Permitir realizar una previsualización de la escena a generar por medio de la herramienta de BMRT, rgl; permitiendo el control sobre los siguientes parámetros: alto y ancho de la imagen, cuadro inicial y cuadro final, cuadros por segundo, nivel de detalle y estilo de dibujado (normal, líneas y mano alzada)

29. Permitir la generación final de la escena usando la herramienta de BMRT, rendrib, especificando los siguientes parámetros: alto y ancho de la imagen, cuadro inicial y final de la animación, número de pixeles de muestreo por barrido horizontal y vertical, tipo de procesamiento del render (local o en red), ventana de recorte (crop), número de pasos de previsualización (preview steps), valores de radiosidad (pasos y número de rayos por pixel), tipo de

salida que se desea (tiff, MPEG-1 y secuencia de tiff) y el nombre de el (los) archivo(s) a generar si así se desea.

30. Permitir la visualización del último archivo de imagen o vídeo generado.

El Módulo de almacenamiento de escenas debe estar en capacidad de:

31. Abrir una escena definida en el formato xrib (formato propio del programa)

32. Grabar una escena en los siguientes formatos xrib.

33. Cerrar la escena actual y preguntar si desea grabarla.

34. Crear una nueva escena, lo que implica guardar la actual y crear una nueva en blanco.

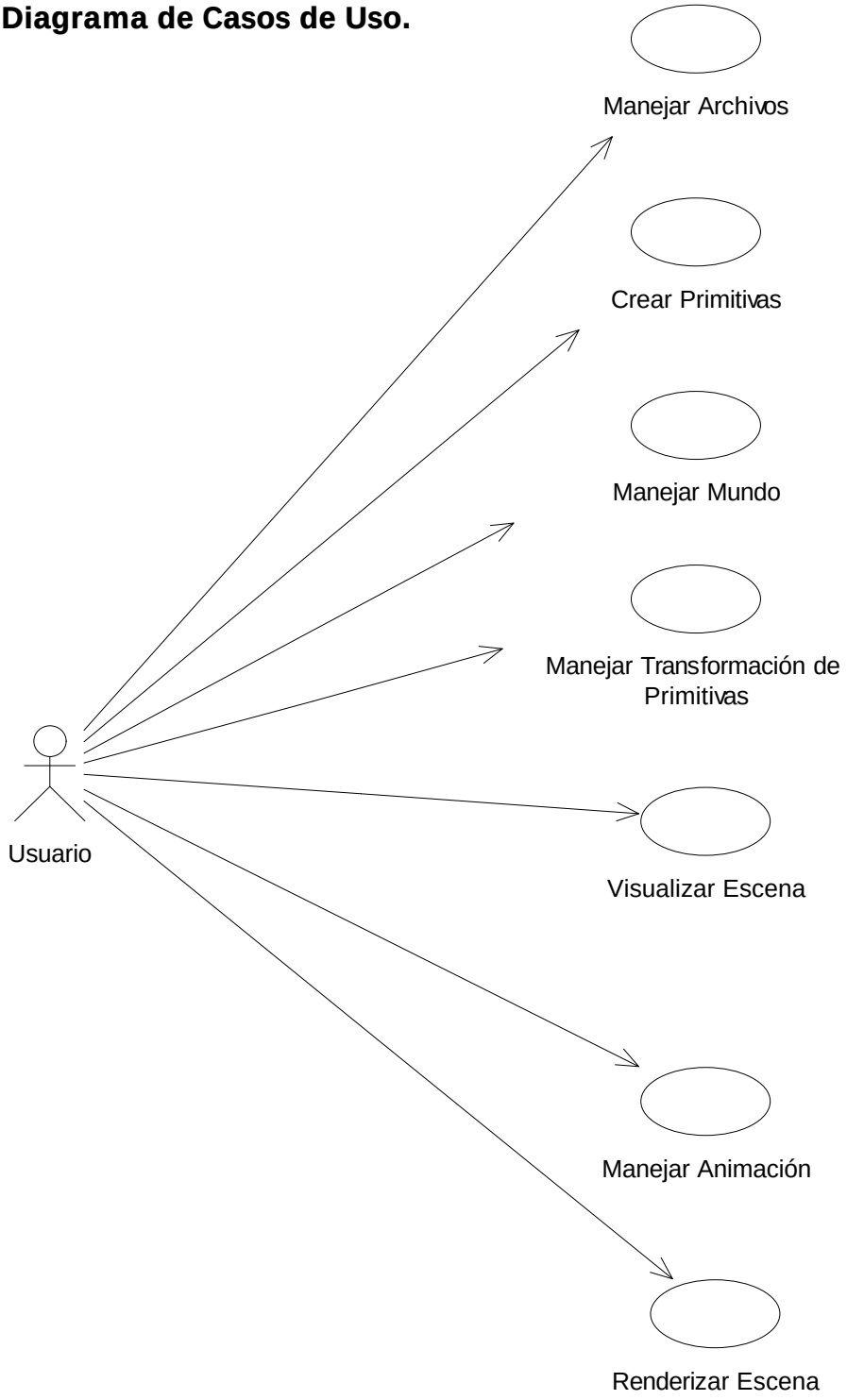
35. Guardar la escena como formato rib (guardar como).

5.1.2. Requerimientos no funcionales

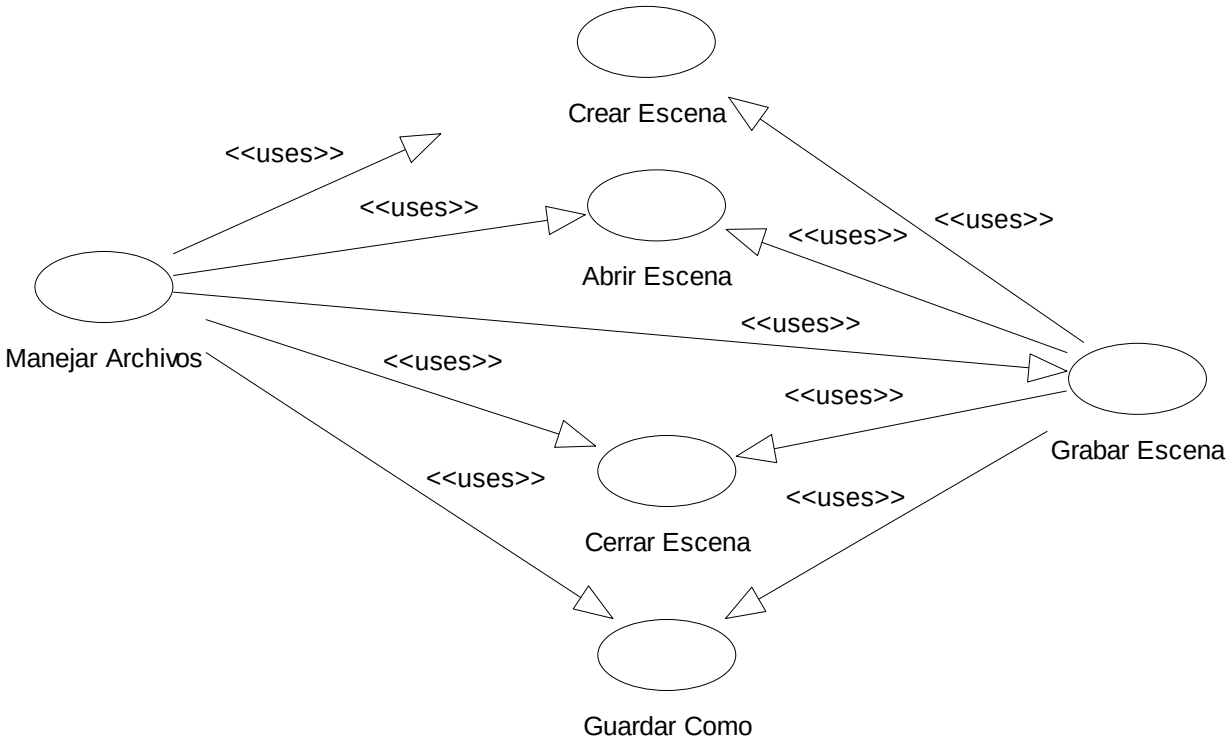
1. Desarrollar e implementar una interfaz gráfica de usuario (GUI) en C++ mediante el API (Aplication Programing Interface) Qt que permita el acceso a los módulos del programa.
2. Análisis y diseño de la herramienta de software en UML
3. Realización del manual de usuario del programa el cual estará en formato HTML.

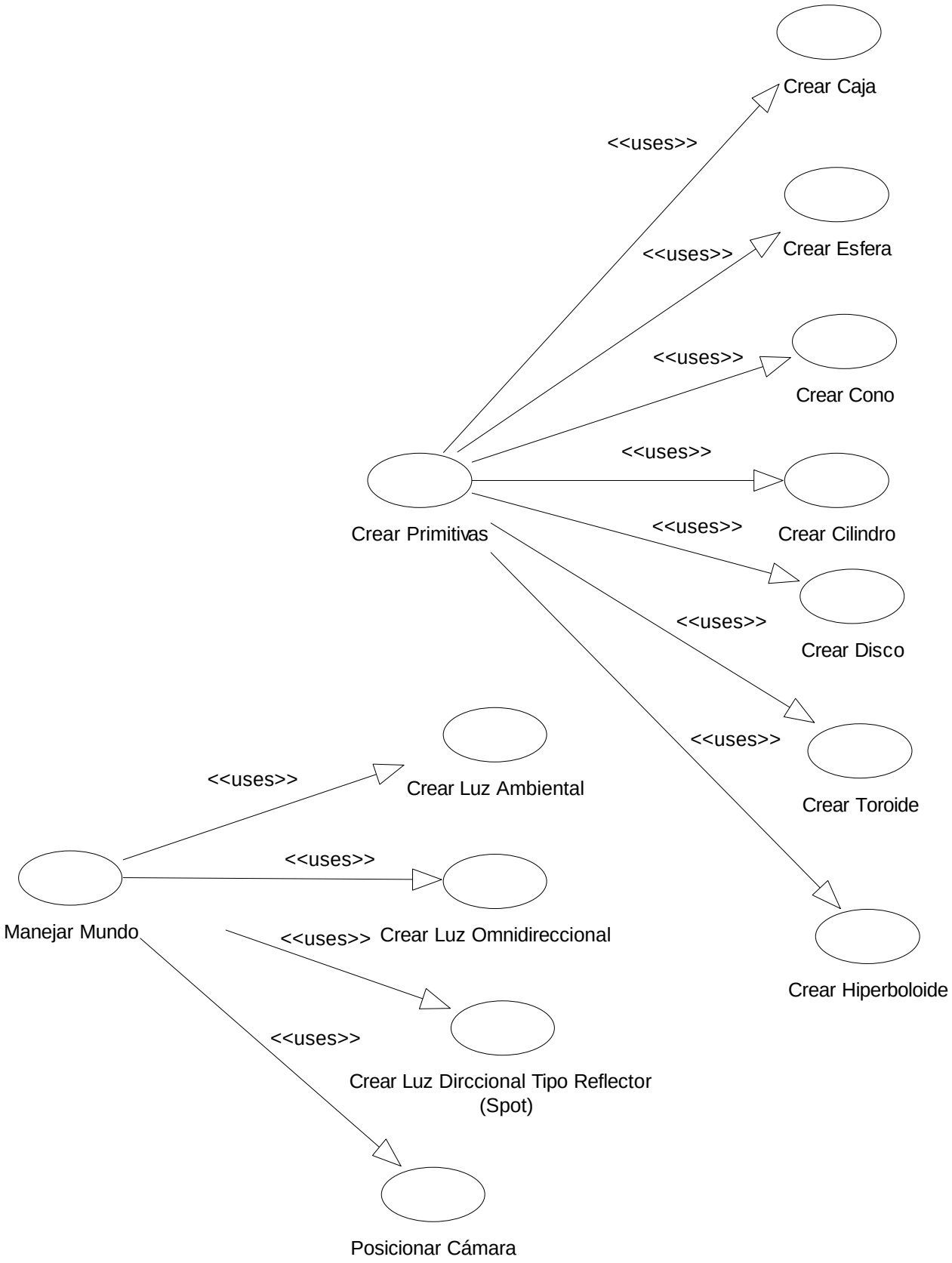
5.2 Definición de los Casos de Uso

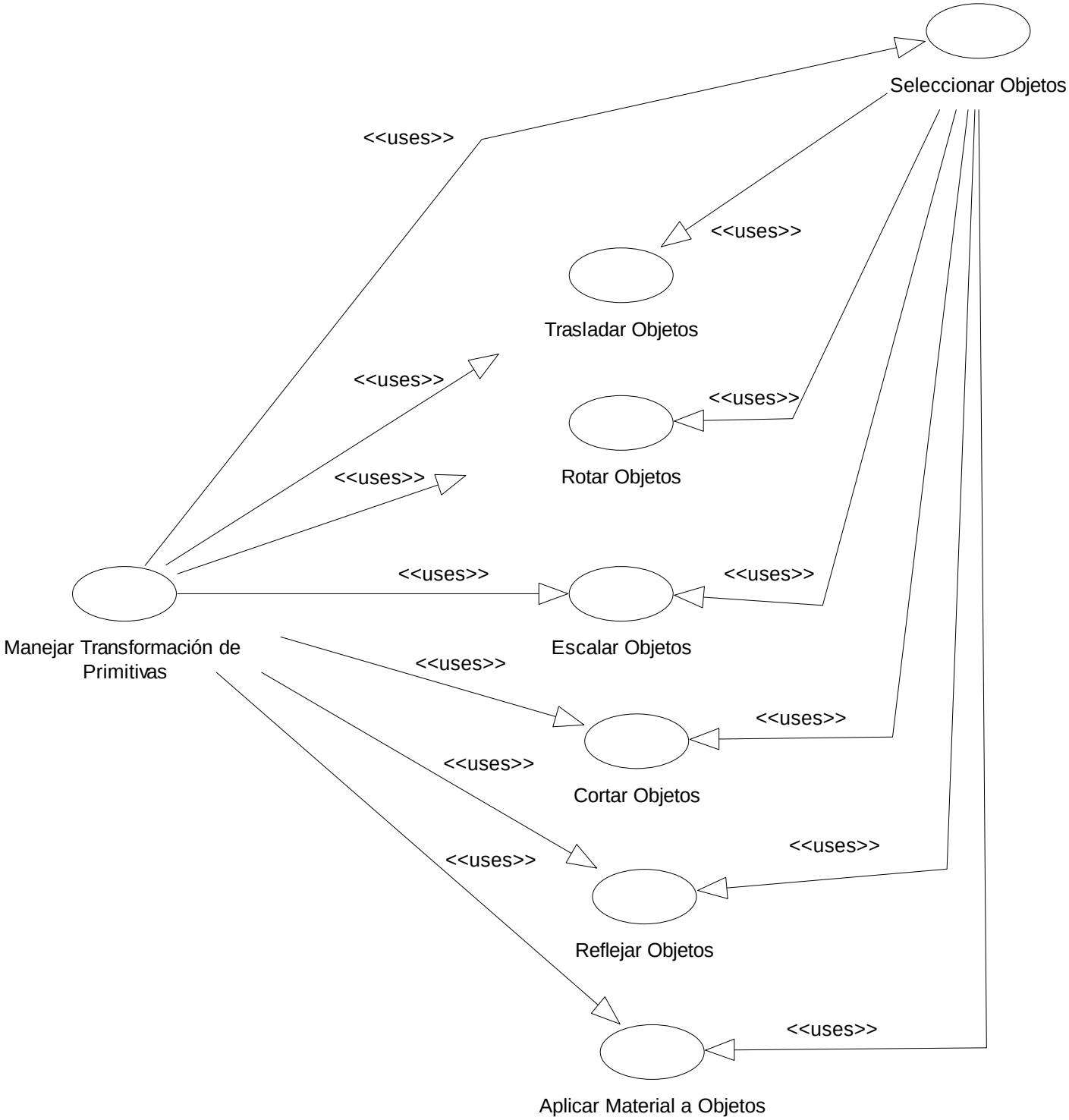
5.2.1 Diagrama de Casos de Uso.

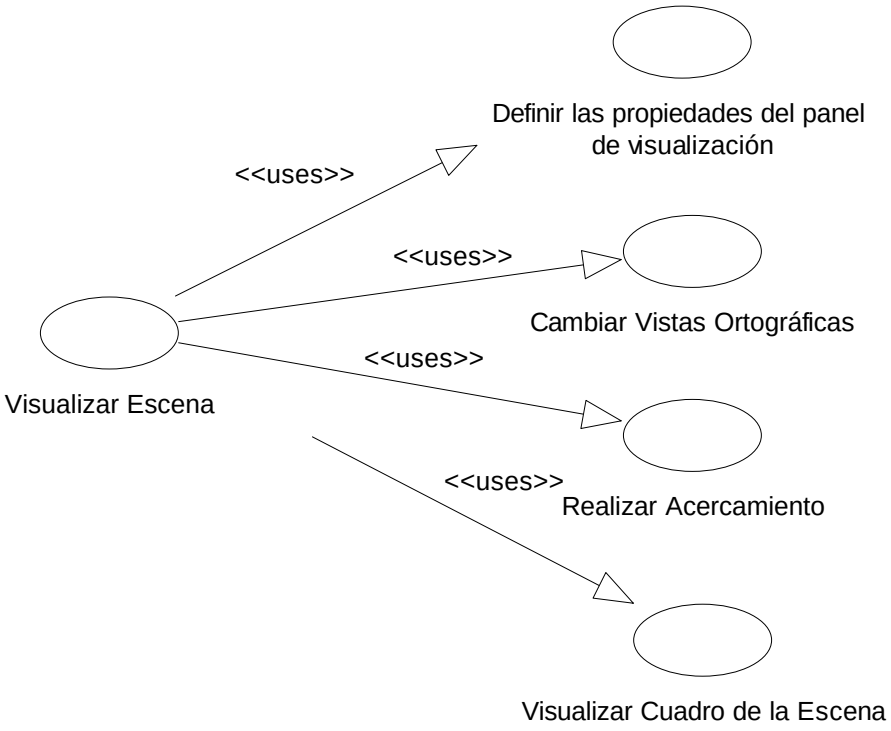


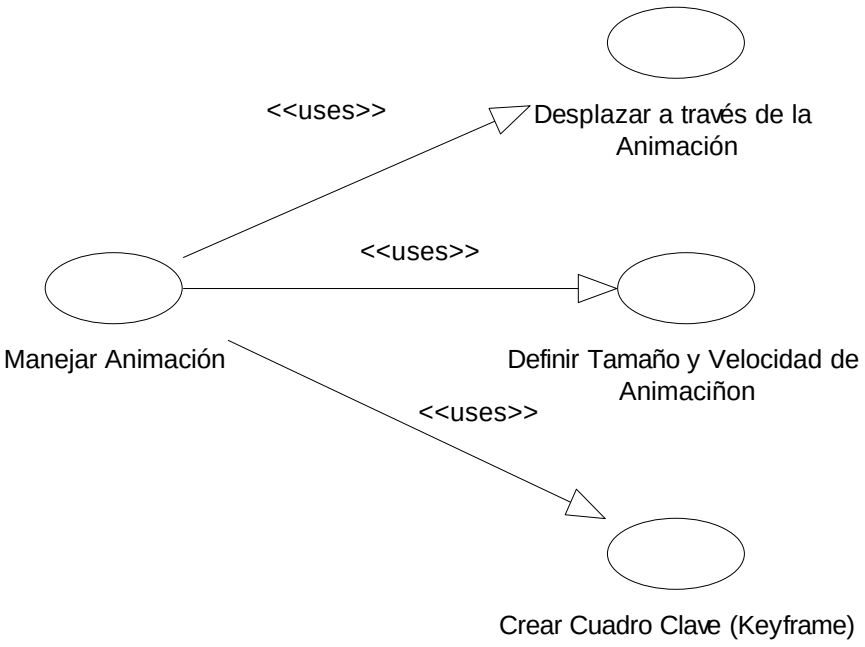
5.2.1.1 Diagramas de Casos de Uso extendidos




Crear Caja
<<uses>>
Crear Esfera
<<uses>>
Crear Cono
<<uses>>
<<uses>>
Crear Primitivas
Crear Cilindro
<<uses>>
Crear Disco
<<uses>>
Crear Toroide
<<uses>>
Crear Hiperboloide
Crear Luz Ambiental
<<uses>>
<<uses>>
Manejar Mundo
Crear Luz Omnidireccional
<<uses>>
<<uses>>
Crear Luz Dirccional Tipo Reflector
(Spot)
Posicionar Cámara


Seleccionar Objetos
<<uses>>
<<uses>>
Trasladar Objetos
<<uses>>
<<uses>>
Rotar Objetos
<<uses>>
<<uses>>
<<uses>>
Manejar Transformación de Primitivas
Escalar Objetos
<<uses>>
<<uses>>
Cortar Objetos
<<uses>>
<<uses>>
Reflejar Objetos
<<uses>>
Aplicar Material a Objetos


Definir las propiedades del panel de visualización
<<uses>>
<<uses>>
Cambiar Vistas Ortográficas
<<uses>>
Visualizar Escena
<<uses>>
Realizar Acercamiento
Visualizar Cuadro de la Escena


<<uses>>
Desplazar a través de la Animación
<<uses>>
Manejar Animación
Definir Tamaño y Velocidad de Animaciñon
<<uses>>
Crear Cuadro Clave (Keyframe)

5.2.2. Documentación de los casos de uso

Nombre	Manejar archivos
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema acceder a las funciones de almacenamiento y recuperación de las escenas.
Tipo	Primario y esencial
Propósito	Permitir al usuario acceder a las funciones de creación, carga, almacenamiento y cerrado de una escena.
Referencias Cruzadas	Requerimientos funcionales: 1, 31, 32, 33, 34, 35. Casos de uso: Crear escena, Abrir escena, Grabar escena, Cerrar escena, Guardar Como.

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario inicia la aplicación.	2. El sistema despliega la interfaz gráfica de usuario.
3. El usuario va al modulo de manejo de archivos (scene)	4. La aplicación despliega un a barra de tareas con las opciones de: nueva escena, grabar escena, graba como y cerrar escena.
5. El usuario selecciona cualquiera de las opciones desplegadas iniciando así otro caso de uso.	

Nombre	Grabar Escena
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema almacenar la escena actual.
Tipo	Primario y esencial
Propósito	Permitir al usuario el almacenamiento de la escena en el formato xrib.
Referencias cruzadas	Requerimientos funcionales: 32 Casos de uso: Manejar Archivos (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón grabar escena (save)	2. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario el nombre del archivo donde desea almacenar la escena.
3. El usuario ingresa el nombre del archivo y presiona Guardar.	4. El sistema almacena la escena actual en el archivo especificado por el usuario.

Nombre	Crear Escena
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear una nueva escena.
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de una nueva escena consultando al usuario si desea grabar la actual.
Referencias cruzadas	Requerimientos funcionales: 34 Casos de uso: Manejar Archivos (<<uses>>), Grabar escena (<< uses >>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear escena (new)	2. El sistema despliega un cuadro de dialogo el cual le preguntará al usuario si desea guardar la escena actual.
3. El usuario presiona no.	4. El sistema reinicializa el estado de los gráficos y de las estructuras de almacenamiento internas.
	5. El sistema despliega el módulo de visualización completamente reinicializado.

Curso alterno de los eventos

3.a El usuario presiona si.
4.a El sistema usa el caso de uso grabar escena, partiendo del paso 2 de éste y continúa con el curso normal a partir del paso 4.

Nombre	Abir escena
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema cargar una escena definida en el formato xrib.
Tipo	Primario y esencial
Propósito	Permitir al usuario la carga de un archivo
Referencias cruzadas	Requerimientos funcionales: 31 Casos de uso: Manejar Archivos (<<uses>>), Grabar escena (<< uses >>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón abrir escena (open)	2. El sistema despliega un cuadro de dialogo el cual le preguntará al usuario el archivo xrib a cargar.
3. El usuario selecciona el archivo y presiona ok.	4. El sistema despliega un dialogo preguntando si desea guardar la escena actual.
5. El usuario responde no.	6. El sistema reinicializa el estado de los gráficos y de las estructuras de almacenamiento internas.
	7. El sistema procede a la carga de las estructuras y el estado de los gráficos consignados en el archivo a cargar.
	8. El sistema realiza el despliegue visual de la escena cargada.

Curso alterno de los eventos

5.a El usuario presiona si.
6.a El sistema usa el caso de uso grabar escena, partiendo del paso 2 de éste y continúa con el curso normal a partir del paso 6.

Nombre	Cerrar escena
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema cerrar la escena actual.
Tipo	Primario y esencial
Propósito	Permitir al usuario cerrar la escena
Referencias cruzadas	Requerimientos funcionales: 33 Casos de uso: Manejar Archivos (<<uses>>), Grabar escena (<< uses >>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón cerrar escena (close)	2. El sistema despliega un cuadro de dialogo el cual le preguntará al usuario si desea grabar la escena actual.
3. El usuario responde no.	4. El sistema reinicializa el estado de los gráficos y de las estructuras de almacenamiento internas.
	5. El sistema despliega el módulo de visualización completamente reinicializado

Curso alterno de los eventos

3.a El usuario presiona si.
4.a El sistema usa el caso de uso grabar escena, partiendo del paso 2 de éste y continúa con el curso normal a partir del paso 4.

Nombre	Guardar como
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema exportar la escena actual a formato rib.
Tipo	Primario y esencial
Propósito	Permitir al usuario grabar la escena en formato rib.
Referencias cruzadas	Requerimientos funcionales: 35 Casos de uso: Manejar Archivos (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón grabar como (save as)	2. El sistema despliega un cuadro de dialogo el cual le preguntará al usuario el nombre del archivo rib al cual desea grabar la escena actual.
3. El usuario ingresa el nombre del archivo.	4. El sistema almacena la escena en el formato rib al archivo especificado.

Nombre	Crear Primitivas
Actores	Usuario
Resumen	Permitir al usuario acceder al módulo de creación de primitivas.
Tipo	Primario y esencial
Propósito	Este caso de uso le permite a los usuarios del sistema acceder a las funciones de creación de las primitivas básicas: caja, esfera, cono, cilindro, disco, toroide, hiperboloide.
Referencias cruzadas	Requerimientos funcionales: 1, 2,3,4,5,6,7,8. Casos de uso: crear caja, crear esfera, crear cono, crear cilindro, crear disco, crear toroide, crear hiperboloide.

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario inicia la aplicación.	2. El sistema despliega la interfaz gráfica de usuario.
3. El usuario va al modulo de creación de primitivas (objects)	4. La aplicación despliega un a barra de tareas con las opciones de: crear caja, esfera, cono, cilindro, disco, toroide, hiperboloide.
5. El usuario selecciona cualquiera de las opciones desplegadas iniciando así otro caso de uso.	

Nombre	Crear Caja
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear una primitiva tipo prisma rectangular (caja)
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de una primitiva tipo prisma rectangular definiendo parámetros de nombre, color, alto, ancho y profundidad demás de posición en el espacio
Referencias cruzadas	Requerimientos funcionales: 2 Casos de uso: Crear primitivas (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear caja (box)	2. El sistema crea una primitiva temporal con valores por defecto.
	3. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: nombre de la caja, color, alto, ancho, profundidad, posición x,y,z.
4. El usuario ingresa los datos solicitados y presiona crear (create)	5. Conforme el usuario modifica los datos, el sistema se encarga de actualizar la visualización del objeto según los nuevos parámetros, para posteriormente crearlo al recibir la señal indicada.
	6. El sistema se encarga de agregar la primitiva creada a su lista de primitivas.

Curso alterno de los eventos

4.a El usuario presiona cancelar (cancel)

5.a El sistema libera la memoria utilizada para la creación de la primitiva temporal.

Nombre	Crear Esfera
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear una primitiva tipo esfera (sphere)
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de una primitiva tipo esfera definiendo parámetros de nombre, color, radio, zmax, zmin, y ángulo theta.
Referencias cruzadas	Requerimientos funcionales: 3 Casos de uso: Crear primitivas (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear esfera (sphere)	2. El sistema crea una primitiva temporal con valores por defecto.
	3. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: nombre, color, radio, zmax, zmin, y ángulo theta.
4. El usuario ingresa los datos solicitados y presiona crear (create)	5. Conforme el usuario modifica los datos, el sistema se encarga de actualizar la visualización del objeto según los nuevos parámetros, para posteriormente crearlo al recibir la señal indicada.
	6. El sistema se encarga de agregar la primitiva creada a su lista de primitivas.

Curso alterno de los eventos

4.a El usuario presiona cancelar (cancel)

5.a El sistema libera la memoria utilizada para la creación de la primitiva temporal.

Nombre	Crear Cono
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear una primitiva tipo conoidal (cone)
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de una primitiva tipo esfera definiendo parámetros de nombre, color, altura, ángulo theta y radio.
Referencias cruzadas	Requerimientos funcionales: 4 Casos de uso: Crear primitivas (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear cono (cone)	2. El sistema crea una primitiva temporal con valores por defecto.
	3. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: nombre, color, altura, ángulo theta y radio.
4. El usuario ingresa los datos solicitados y presiona crear (create)	5. Conforme el usuario modifica los datos, el sistema se encarga de actualizar la visualización del objeto según los nuevos parámetros, para posteriormente crearlo al recibir la señal indicada.
	6. El sistema se encarga de agregar la primitiva creada a su lista de primitivas.

Curso alterno de los eventos

4.a El usuario presiona cancelar (cancel)

5.a El sistema libera la memoria utilizada para la creación de la primitiva temporal.

Nombre	Crear Cilindro
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear una primitiva tipo cilindro (cylinder)
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de una primitiva tipo esfera definiendo parámetros de nombre, color, zmax, zmin, radio, ángulo theta
Referencias cruzadas	Requerimientos funcionales: 5 Casos de uso: Crear primitivas (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear cilindro (cylinder)	2. El sistema crea una primitiva temporal con valores por defecto.
	3. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: nombre, color, zmax, zmin, radio, ángulo theta
4. El usuario ingresa los datos solicitados y presiona crear (create)	5. Conforme el usuario modifica los datos, el sistema se encarga de actualizar la visualización del objeto según los nuevos parámetros, para posteriormente crearlo al recibir la señal indicada.
	6. El sistema se encarga de agregar la primitiva creada a su lista de primitivas.

Curso alterno de los eventos

4.a El usuario presiona cancelar (cancel)

5.a El sistema libera la memoria utilizada para la creación de la primitiva temporal.

Nombre	Crear Disco
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear una primitiva tipo disco (disk)
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de una primitiva tipo esfera definiendo parámetros de nombre, color, radio, altura y ángulo theta
Referencias cruzadas	Requerimientos funcionales: 6 Casos de uso: Crear primitivas (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear disco (disk)	2. El sistema crea una primitiva temporal con valores por defecto.
	3. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: nombre, color, radio, altura y ángulo theta
4. El usuario ingresa los datos solicitados y presiona crear (create)	5. Conforme el usuario modifica los datos, el sistema se encarga de actualizar la visualización del objeto según los nuevos parámetros, para posteriormente crearlo al recibir la señal indicada.
	6. El sistema se encarga de agregar la primitiva creada a su lista de primitivas.

Curso alterno de los eventos

4.a El usuario presiona cancelar (cancel)
5.a El sistema libera la memoria utilizada para la creación de la primitiva temporal.

Nombre	Crear Toroide
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear una primitiva tipo toroidal (torus)
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de una primitiva tipo esfera definiendo parámetros de nombre, color, radio mayor, radio menor, ángulo phi mayor, ángulo phi menor y ángulo theta
Referencias cruzadas	Requerimientos funcionales: 7 Casos de uso: Crear primitivas (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear toroide (torus)	2. El sistema crea una primitiva temporal con valores por defecto.
	3. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: nombre, color, radio mayor, radio menor, ángulo phi mayor, ángulo phi menor y ángulo theta
4. El usuario ingresa los datos solicitados y presiona crear (create)	5. Conforme el usuario modifica los datos, el sistema se encarga de actualizar la visualización del objeto según los nuevos parámetros, para crearlo al recibir la señal indicada.
	6. El sistema se encarga de agregar la primitiva creada a su lista de primitivas.

Curso alterno de los eventos

4.a El usuario presiona cancelar (cancel)
5.a El sistema libera la memoria utilizada para la creación de la primitiva temporal.

Nombre	Crear Hiperboloide
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear una primitiva tipo Hiperbola Tridimensional (Hyperboloid)
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de una primitiva tipo esfera definiendo parámetros de nombre, color, punto inicial (x,y,z), punto final (x1,y1,z1) y ángulo entre puntos.
Referencias cruzadas	Requerimientos funcionales: 8 Casos de uso: Crear primitivas (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear hiperboloide (hyperboloid)	2. El sistema crea una primitiva temporal con valores por defecto.
	3. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: nombre, color, punto inicial (x,y,z), punto final (x1,y1,z1) y ángulo entre puntos.
4. El usuario ingresa los datos solicitados y presiona crear (create)	5. Conforme el usuario modifica los datos, el sistema se encarga de actualizar la visualización del objeto según los nuevos parámetros, para posteriormente crearlo al recibir la señal indicada.
	6. El sistema se encarga de agregar la primitiva creada a su lista de primitivas.

Curso alterno de los eventos

4.a El usuario presiona cancelar (cancel)
5.a El sistema libera la memoria utilizada para la creación de la primitiva temporal.

Nombre	Manejar Mundo
Actores	Usuario
Resumen	Permitir al usuario acceder al módulo de creación de luces y modificación de cámara
Tipo	Primario y esencial
Propósito	Este caso de uso le permite a los usuarios del sistema acceder al manejo de las luces ambiental, omnidireccional y tipo reflector y de la cámara
Referencias	Requerimientos funcionales: 1,9,10,11,12

cruzadas Casos de uso: crear luz ambiental , crear luz omnidiereccional, crear luz tipo reflector , posicionar cámara

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario inicia la aplicación.	2. El sistema despliega la interfaz gráfica de usuario.
3. El usuario va al modulo de manejo de luces y cámara (word)	4. La aplicación despliega un a barra de tareas con las opciones de: crear luz ambiental, crear luz omnidiereccional, crear luz tipo reflector, posicionar cámara.
5. El usuario selecciona cualquiera de las opciones desplegadas iniciando así otro caso de uso.	

Nombre Crear Luz Ambiental

Actores Usuario

Resumen Este caso de uso le permite a los usuarios del sistema definir los parámetros de la luz ambiental para la escena en general

Tipo Primario y esencial

Propósito Permitir al usuario definir la luz ambiental para la escena

Referencias cruzadas Requerimientos funcionales: 9

Casos de uso: Manejar Mundo(<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear luz ambiental (ambient ligth)	2. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: intensidad y color
3. El usuario ingresa los datos solicitados y presiona ok	4. El sistema se encarga de almacenar los valores de la luz ambiental como una cadena en una lista de cadenas que expresan los valores de las luces

Curso alterno de los eventos

3.a El usuario presiona cancelar (cancel)
4.a El sistema cierra el dialogo y no almacena ningún dato.

Nombre	Crear Luz Omnidireccional
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear luces omnidireccionales.
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de luces omnidireccionales especificando los parámetros de intensidad, color y posición en el espacio.
Referencias cruzadas	Requerimientos funcionales: 10 Casos de uso: Manejar Mundo(<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear luz omnidireccional (omni ligth)	2. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: intensidad, color y posición x,y,z.
3. El usuario ingresa los datos solicitados y presiona ok	4. Conforme el usuario ingresa los datos, el sistema despliega un punto que representa la posición de la luz y posteriormente se encarga de almacenar los parámetros de ésta en una cadena que se insertará en la lista de cadenas de luces.

Curso alterno de los eventos

3.a El usuario presiona cancelar (cancel)
4.a El sistema cierra el dialogo y no almacena ningún dato.

Nombre	Crear Luz Direccional Tipo Reflector
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema crear luces direccionales tipo reflector
Tipo	Primario y esencial
Propósito	Permitir al usuario la creación de luces direccionales tipo reflector especificando los parámetros de intensidad, color y posición en el espacio origen y destino, ángulo del cono, exponente de decaimiento y ángulo de decaimiento
Referencias cruzadas	Requerimientos funcionales: 11 Casos de uso: Manejar Mundo(<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón crear luces direccionales tipo reflector (spot ligth)	2. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: intensidad, color y posición en el espacio origen y destino, ángulo del cono, exponente de decaimiento y ángulo de decaimiento
3. El usuario ingresa los datos solicitados y presiona ok	4. Conforme el usuario ingresa los datos, el sistema despliega una línea que representa desde y hasta dónde la luz afecta los objetos y posteriormente se encarga de almacenar los parámetros de ésta en una cadena que se insertará en la lista de cadenas de luces.

Curso alterno de los eventos

3.a El usuario presiona cancelar (cancel)
4.a El sistema cierra el dialogo y no almacena ningún dato.

Nombre	Posicionar Cámara
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema definir la posición de la cámara
Tipo	Primario y esencial
Propósito	Permitir al usuario la definición de la cámara, especificando los valores de posición ésta y hacia dónde se encuentra mirando.
Referencias cruzadas	Requerimientos funcionales: 12 Casos de uso: Manejar Mundo(<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón posicionar cámara (camera)	2. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: posición origen x,y,z y posición destino x1,y1,z1
3. El usuario ingresa los datos solicitados y presiona ok	4. Conforme el usuario ingresa los datos, el sistema muestra los cambios de la cámara y posteriormente almacena los valores ingresados en el último keyframe activo

Curso alterno de los eventos

3.a El usuario presiona cancelar (cancel)
4.a El sistema cierra el dialogo y no almacena ningún dato y retorna la cámara a su posición original.

Nombre	Manejar Transformaciones de Primitivas
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema acceder a las funciones de modificación de los objetos de la escena
Tipo	Primario y esencial
Propósito	Permitir al usuario acceder a las funciones de: seleccionar, trasladar, rotar, escalar, cortar, reflejar y aplicar materiales a objetos
Referencias	Requerimientos funcionales: 1, 13,14,15,16,17,18,19
Cruzadas	Casos de uso: Seleccionar Objetos, trasladar objetos, rotar objetos, escalar objetos, cortar objetos, reflejar objetos, aplicar material a objetos.

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario inicia la aplicación.	2. El sistema despliega la interfaz gráfica de usuario.
3. El usuario va al Manejo de Transformación de Primitivas	4. La aplicación despliega un a barra de tareas con las opciones de: Seleccionar Objetos, trasladar objetos, rotar objetos, escalar objetos, cortar objetos, reflejar objetos, aplicar material a objetos
5. El usuario selecciona cualquiera de las opciones desplegadas iniciando así otro caso de uso.	

Nombre	Seleccionar Objetos
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema seleccionar objetos por nombre
Tipo	Primario y esencial
Propósito	Permitir al usuario la selección de uno o múltiples objetos por medio de un cuadro de selección con los nombres de las primitivas
Referencias cruzadas	Requerimientos funcionales: 13 Casos de uso: Manejar Transformaciones de primitivas (<<uses>>), trasladar objetos, rotar objetos, escalar objetos, cortar objetos, reflejar objetos, aplicar material a objetos.

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón Seleccionar Objetos (Selection)	2. El sistema despliega un cuadro de dialogo en el cual muestra los objetos actualmente creados en la escena los cuales pueden ser seleccionados para una posterior modificación
3. El usuario selecciona los objetos que desea modificar	4. Conforme el usuario selecciona los objetos, estos toman un color blanco para indicar que se encuentran seleccionados
5. El usuario presiona ok	6. El sistema se encarga de activar la bandera seleccionado, a cada uno de los objetos seleccionados por el usuario.

Curso alterno de los eventos

5.a El usuario presiona cancelar (cancel)
6.a El sistema cierra el dialogo, no almacena ningún dato y retorna los objetos a su color original.

Nombre	Trasladar Objetos
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema trasladar los objetos
Tipo	Primario y esencial
Propósito	Permitir al usuario la traslación en el espacio de los objetos seleccionados.
Referencias cruzadas	Requerimientos funcionales: 14 Casos de uso: Manejar Transformaciones de primitivas (<<uses>>), Seleccionar Objetos (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón Trasladar Objetos (Translate)	2. El sistema despliega un cuadro de dialogo el cual le solicita al usuario las cantidades en x, y, y z que desea trasladar los objetos.
3. El usuario ingresa los valores de la traslación	4. Conforme el usuario modifica los valores de x, y, y z el sistema muestra la nueva posición de los objetos en el espacio actualizando la matriz de transformación de visualización y multiplicándosela a los objetos seleccionados.
5. El usuario presiona ok	6. El sistema crea una matriz de transformación con los valores ingresados por el usuario y posteriormente la concatena a la matriz de transformación actual de cada primitiva.

Curso alterno de los eventos

5.a El usuario presiona cancelar (cancel)
6.a El sistema cierra el dialogo, no almacena ningún dato y retorna la matriz de transformación de visualización a su valor original.

Nombre	Rotar Objetos
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema rotar los objetos
Tipo	Primario y esencial
Propósito	Permitir al usuario la rotación en el espacio de los objetos seleccionados.
Referencias cruzadas	Requerimientos funcionales: 15 Casos de uso: Manejar Transformaciones de primitivas (<<uses>>), Seleccionar Objetos (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón Rotar Objetos (Rotate)	2. El sistema despliega un cuadro de dialogo el cual le solicita al usuario el eje axial y el ángulo sobre el cual aplicará la rotación
3. El usuario ingresa los valores de la rotación	4. Conforme el usuario modifica los valores el sistema muestra la nueva rotación de los objetos en el espacio actualizando la matriz de transformación de visualización y multiplicándosela a los objetos seleccionados.
5. El usuario presiona ok	6. El sistema crea una matriz de transformación con los valores ingresados por el usuario y posteriormente la concatena a la matriz de transformación actual de cada primitiva.

Curso alterno de los eventos

5.a El usuario presiona cancelar (cancel)
6.a El sistema cierra el dialogo, no almacena ningún dato y retorna la matriz de transformación de visualización a su valor original.

Nombre	Escalar Objetos
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema escalar los objetos
Tipo	Primario y esencial
Propósito	Permitir al usuario el escalado en el espacio de los objetos seleccionados respecto a los ejes.
Referencias cruzadas	Requerimientos funcionales: 16 Casos de uso: Manejar Transformaciones de primitivas

(<<uses>>), Seleccionar Objetos (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón Escalar Objetos (Scale)	2. El sistema despliega un cuadro de dialogo el cual le solicita al usuario los factores de escalado para cada eje
3. El usuario ingresa los valores del escalado	4. Conforme el usuario modifica los valores el sistema muestra los objetos con su nuevo tamaño en el espacio actualizando la matriz de transformación de visualización y multiplicándosela a los objetos seleccionados.
5. El usuario presiona ok	6. El sistema crea una matriz de transformación con los valores ingresados por el usuario y posteriormente la concatena a la matriz de transformación actual de cada primitiva.

Curso alterno de los eventos

5.a El usuario presiona cancelar (cancel)
6.a El sistema cierra el dialogo, no almacena ningún dato y retorna la matriz de transformación de visualización a su valor original.

Nombre	Cortar Objetos
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema aplicar operaciones de corte a los objetos seleccionados
Tipo	Primario y esencial
Propósito	Permitir al usuario el realizar operaciones de corte respecto a un eje axial.
Referencias cruzadas	Requerimientos funcionales: 17 Casos de uso: Manejar Transformaciones de primitivas

(<<uses>>), Seleccionar Objetos (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón Cortar Objetos (Share)	2. El sistema despliega un cuadro de dialogo el cual le solicita al usuario el eje sobre el cual se realizará el corte y los valores a y b de corte respecto a los demás ejes.
3. El usuario ingresa los valores del corte	4. Conforme el usuario modifica los valores el sistema muestra los objetos con el respectivo corte actualizando la matriz de transformación de visualización y multiplicándosela a los objetos seleccionados.
5. El usuario presiona ok	6. El sistema crea una matriz de transformación con los valores ingresados por el usuario y posteriormente la concatena a la matriz de transformación actual de cada primitiva.

Curso alterno de los eventos

5.a El usuario presiona cancelar (cancel)
6.a El sistema cierra el dialogo, no almacena ningún dato y retorna la matriz de transformación de visualización a su valor original.

Nombre	Reflejar Objetos
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema aplicar operaciones de reflejado a los objetos seleccionados
Tipo	Primario y esencial
Propósito	Permitir al usuario el realizar operaciones de reflejado respecto a los ejes axial.
Referencias	Requerimientos funcionales: 17

cruzadas Casos de uso: Manejar Transformaciones de primitivas (<<uses>>), Seleccionar Objetos (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón Reflejar Objetos (Mirror)	2. El sistema despliega un cuadro de dialogo el cual le solicita los ejes sobre los cuales desea realizar la reflexión
3. El usuario ingresa los valores del corte	4. Conforme el usuario modifica los valores el sistema muestra los objetos reflejados sobre los ejes seleccionados actualizando la matriz de transformación de visualización y multiplicándosela a los objetos seleccionados.
5. El usuario presiona ok	6. El sistema crea una matriz de transformación con los valores ingresados por el usuario y posteriormente la concatena a la matriz de transformación actual de cada primitiva.

Curso alterno de los eventos

5.a El usuario presiona cancelar (cancel)
6.a El sistema cierra el dialogo, no almacena ningún dato y retorna la matriz de transformación de visualización a su valor original.

Nombre Aplicar Material a Objetos

Actores Usuario

Resumen Este caso de uso le permite a los usuarios del sistema aplicar materiales a los objetos seleccionados

Tipo Primario y esencial

Propósito Permitir al usuario aplicar materiales de la librería de materiales a los objetos seleccionados, indicando los valores de coeficiente ambiental, difuso y especular,

además del color del objeto y de la luz especular.

Referencias cruzadas Requerimientos funcionales: 19

Casos de uso: Manejar Transformaciones de primitivas (<<uses>>), Seleccionar Objetos (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón Asignar Material a Objetos (Material)	2. El sistema despliega un cuadro de dialogo el cual contiene un listado de los materiales disponibles proveídos por Xrib además de los valores de coeficiente ambiental, difuso, especular, color del objeto y color de la luz especular.
3. El usuario selecciona el material a aplicar y los valores de coeficiente ambiental, difuso, especular, el color del objeto y el color de la luz especular y presiona ok	4. El sistema almacena los valores para el material y el color de los objetos , estos cambios sólo pueden ser visualizados durante la generación de la imagen o el vídeo final

Curso alterno de los eventos

3.a El usuario presiona cancelar (cancel)
6.a El sistema cierra el dialogo y no almacena ningún dato.

Nombre Visualizar Escena

Actores Usuario

Resumen Este caso de uso le permite a los usuarios del sistema visualizar por medio de OpenGL las primitivas y luces que conforme se vayan creando en la escena.

Tipo Primario y esencial

Propósito Permitir al usuario visualizar la escena de manera interactiva.

Referencias Cruzadas Requerimientos funcionales: 1,20,21,22,23,24

Casos de uso: Definir propiedades del panel de visualización., Cambiar las vistas ortográficas, Realizar acercamientos, Visualizar cuadro de la escena.

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario inicia la aplicación.	2. El sistema despliega la interfaz gráfica de usuario.
	3. la interfaz crea 4 instancias de los applets de OpenGl (paneles) los cuales quedan a la espera de señales de actualización (repaint) para desplegar de nuevo las listas de gráficos que posea OpenGl en ese momento.

Nombre Definir propiedades del panel de visualización.

Actores Usuario

Resumen Este caso de uso le permite a los usuarios del sistema definir el límite cercano y lejano (near, far) del volumen de vista además del ángulo de vista (fov), el color de fondo del panel y el tipo de despliegue (plano o en estructura de alambre).

Tipo Primario y esencial

Propósito Permitir al usuario la definición de las propiedades del panel de visualización de OpenGl.

Referencias cruzadas Requerimientos funcionales: 21

Casos de uso: Visualizar escena (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario se dirige a la pestaña Propiedades de Vista (Viewport Settings) de un determinado panel de OpenGL	2. El sistema muestra las propiedades de la vista del panel donde el usuario puede cambiar los valores los límites (near y far), el valor del ángulo de vista (fov) (en caso que el panel seleccionado sea el de la cámara), el color de fondo del panel (si el panel seleccionado es la cámara el color de fondo es el que se utilizará como fondo de la imagen a generar) y el tipo de despliegue (plano o maya)
3. El usuario cambia los valores que considere pertinentes y se cambia a la pestaña view o cámara dependiendo el panel donde se encuentre.	4. El sistema actualiza los valores para el panel modificado y emite la señal de actualización del panel (repaint).

Nombre	Cambiar las Vistas Ortográficas.
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema visualizar las primitivas por medio de un volumen de vista ortográfico es decir sin puntos de fuga para mostrar la profundidad.
Tipo	Primario y esencial
Propósito	Permitir al usuario visualizar la escena desde las vistas ortográficas arriba, abajo, izquierda, derecha, frontal y posterior.
Referencias cruzadas	Requerimientos funcionales: 22 Casos de uso: Visualizar escena (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario se dirige a la pestaña Propiedades de Vista (Viewport Settings) de un determinado panel de OpenGL	2. El sistema muestra las propiedades de la vista del panel, aquí se encuentran las opciones del tipo de vista (solamente si son los paneles 1, 2 o 3) donde el usuario puede escoger entre arriba, abajo, izquierda, derecha, frontal y posterior.
3. El usuario selecciona algún tipo de vista y se cambia a la pestaña del panel de OpenGL.	4. El sistema cambia la lista activa de la cámara, es decir, si se estaba mostrando la lista correspondiente abajo y se cambió a derecha el sistema carga la lista correspondiente a derecha.

Nombre	Realizar acercamientos
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema aplicar un factor de acercamiento (zoom) a los paneles de visualización de OpenGL
Tipo	Primario y esencial
Propósito	Permitir al usuario acercarse o alejarse a los objetos en los paneles de visualización de OpenGL.
Referencias cruzadas	Requerimientos funcionales: 23 Casos de uso: Visualizar escena (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario se dirige a la pestaña Propiedades de Vista (Viewport Settings) de un determinado panel de OpenGL. (Si el usuario lo desea puede acceder a esta función bajo la pestaña de transformaciones (modifiers) esto por comodidad del manejo de la interfaz)	2. El sistema muestra las propiedades de la vista del panel, aquí se encuentra una opción acercamiento (zoom) la cual expresa la escala en valor 1:x.
3. El usuario ingresa el valor x de escala y se cambia a la pestaña del panel de OpenGl. (Por medio de la opción en el módulo de transformaciones, que es una barra de desplazamiento (slider) el valor de x se aplicará a todos los paneles por igual)	4. El sistema actualiza el factor de escala para el panel indicado y envía una señal de actualización de la vista (repaint)

Nombre	Visualizar cuadro de la escena.
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema ver el estado de los gráficos y de las primitivas en un cuadro (frame) determinado.
Tipo	Primario y esencial
Propósito	Permitir al usuario visualizar un cuadro dado de la escena.
Referencias cruzadas	Requerimientos funcionales: 24 Casos de uso: Visualizar escena (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario especifica el cuadro (frame) al cual desea dirigirse, ya sea por medio de la barra de desplazamiento del módulo de animación o especificando el valor en el espacio del número de cuadro actual.	2. El sistema emite una señal cambiarAFrame la cual le avisa a las primitivas y a la cámara que el frame actual es el especificado por el usuario, así que estas ponen a apuntar su estado actual a el del keyframe (cuadro clave) determinado, es decir el frame inmediatamente anterior dónde se hayan almacenado valores.
	4. El envía una señal de actualización de a las vistas de OpenGL (repaint)

Nombre	Manejar Animación
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema acceder a las funciones de animación de la aplicación.
Tipo	Primario y esencial
Propósito	Permitir al usuario el uso de las funciones de animación de desplazamiento a través de los cuadros de la escena, definir el número de cuadros y velocidad de despliegue de los paneles de visualización y crear cuadros claves de animación (keyframe)
Referencias Cruzadas	Requerimientos funcionales: 1,25,26,27 Casos de uso: Desplazar a través de la animación, definir tamaño y velocidad de animación, Crear cuadro clave.

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario inicia la aplicación.	2. El sistema despliega la interfaz gráfica de usuario.
	3. la interfaz inicia la barra de desplazamiento con 100 cuadros y crea un primer cuadro clave 0 para la cámara.

Nombre	Desplazar a través de la animación
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema desplazarse a través de los cuadros de la animación.
Tipo	Primario y esencial
Propósito	Permitir al usuario adelantar, retroceder, avanzar y parar en la animación.
Referencias cruzadas	Requerimientos funcionales: 25 Casos de uso: Manejar Animación (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario selecciona alguna de las opciones de los controles de animación (atrasar, adelantar, avanzar y parar)	2. El sistema envía la señal de cambio de cuadro actual, señal que es recibida por el módulo de visualización el cual se encarga de realizar el despliegue del frame actual en el panel de OpenGl.
	3. En caso que la opción sea avanzar se realiza un ciclo infinito de cambio cuadro a cuadro teniendo en cuenta la velocidad de despliegue para así enviar una señal asíncrona de cambio de frame cada determinado número de milisegundos.

Nombre	Definir tamaño y velocidad de la animación.
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema definir el número de cuadros de la animación y la velocidad con la cual serán desplegados por los paneles de OpenGL.
Tipo	Primario y esencial
Propósito	Permitir al usuario definir el tamaño y velocidad de despliegue de la animación.
Referencias cruzadas	Requerimientos funcionales: 26 Casos de uso: Manejar Animación (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón propiedades de la línea de tiempo (Timeline Settings)	2. El sistema despliega un dialogo que le permite al usuario cambiar los valores del número de cuadros y la velocidad de despliegue.
3. El usuario ingresa los valores para el número de cuadros y la velocidad y presiona Ok.	3. El sistema cambia la longitud de la barra de desplazamiento y del número de cuadros de la escena.

Curso alterno de los eventos

3.a El usuario presiona cancelar (cancel)
4.a El sistema cierra el dialogo y no realiza ningún cambio.

Nombre	Crear Cuadro Clave (Keyframe)
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema la creación de cuadros claves de animación para especificar exactamente en qué escena ocurren

cambios.

Tipo Primario y esencial

Propósito Permitir al usuario la creación de cuadros claves para la animación.

Referencias cruzadas Requerimientos funcionales: 27

Casos de uso: Manejar Animación (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón grabar (Record)	2. El sistema crea un estado para la cámara y lo agrega a la lista de estados de esta indicando el número de frame y le asigna los valores del keyframe inmediatamente anterior. Lo mismo sucede con las primitivas solamente que en lugar de almacenar el keyframe en la lista de estados lo hace en la lista de transformaciones.

Nombre Renderizar Escena

Actores Usuario

Resumen Este caso de uso le permite a los usuarios del sistema acceder a las funciones de generaración de la imagen o vídeo final de la escena, generar una previsualización por medio de OpenGL, y ver el último render realizado.

Tipo Primario y esencial

Propósito Permitir al usuario el uso de las funciones de renderización usando rendrib (programa de BMRT) como algoritmo de render, rgl (programa de BMRT) como previsualizador en OpenGL, iv (programa de BMRT) como visualizador de imágenes tiff y secuencias

de tiff, y generación de videos MPEG-1 por medio de ppmtompeg y tifftoppm (Proveídos en la versión de RedHat Linux 7.0+)

Referencias Cruzadas Requerimientos funcionales: 1,28,29,30

Casos de uso: Previsualizar en OpenGL, Generar imagen o vídeo final, Desplegar último rendering.

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario inicia la aplicación.	2. El sistema despliega la interfaz gráfica de usuario.
3. El usuario va a Renderización de Escenas (Render)	4. La aplicación despliega un a barra de tareas con las opciones de: Render, preview y ver último render (view last rendering)
5. El usuario selecciona cualquiera de las opciones desplegadas iniciando así otro caso de uso.	

Nombre Previsualizar en OpenGl

Actores Usuario

Resumen Este caso de uso le permite a los usuarios del sistema previsualizar el resultado de la escena por medio del programa rgl.

Tipo Primario y esencial

Propósito Permitir al usuario la previsualización del render definiendo parámetros de alto, ancho, desde, hasta, cuadros por segundo, nivel de detalle, y estilo de dibujado (normal, líneas o mano alzada)

Referencias cruzadas Requerimientos funcionales: 28

Casos de uso: Renderizar Escena (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón previsualización (preview)	2. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: alto, ancho, desde, hasta, cuadros por segundo, nivel de detalle, y estilo de dibujado (normal, líneas o mano alzada) y si desea cerrar la ventana al salir.
3. El usuario ingresa los datos solicitados y presiona ok	4. El sistema genera un archivo temporal en formato rib el cual es posteriormente pasado al programa rgl el cual se encarga de desplegar una ventana con la previsualización de la escena según los parámetros ingresados.

Curso alterno de los eventos

3.a El usuario presiona cancelar (cancel)
4.a El sistema cierra el dialogo y no realiza ninguna acción.

Nombre Generar imagen o vídeo final

Actores Usuario

Resumen Este caso de uso le permite a los usuarios del sistema realizar una generación final de la escena en tres formas : imagen tiff, vídeo MPEG-1 o secuencia de tiff de 32 bits con canal alpha, además de habilitar la opción de render en red para el caso de una sola imagen.

Tipo Primario y esencial

Propósito Permitir al usuario la generación final de la escena definiendo los siguientes parámetros:
Alto y ancho de la imagen, cuadro inicial y final de la

	animación, número de pixeles de muestreo por barrido horizontal y vertical, tipo de procesamiento del render (local o en red), ventana de recorte (crop), número de pasos de previsualización (preview steps), valores de radiosidad (pasos y número de rayos por pixel), tipo de salida que se desea (tiff, MPEG-1 y secuencia de tiff) y el nombre del archivo(s) a generar si así se desea.
Referencias cruzadas	Requerimientos funcionales: 29 Casos de uso: Renderizar Escena (<<uses>>), Desplegar último rendering (<<uses>>)

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón render	2. El sistema despliega un cuadro de dialogo en el cual le pregunta al usuario los siguientes parámetros: Alto y ancho de la imagen, cuadro inicial y final de la animación, número de pixeles de muestreo por barrido horizontal y vertical, tipo de procesamiento del render (local o en red), ventana de recorte (crop), número de pasos de previsualización (preview steps), valores de radiosidad (pasos y número de rayos por pixel), tipo de salida que se desea (tiff, MPEG-1 y secuencia de tiff) y el nombre del archivo(s) a generar si así se desea.
3. El usuario ingresa los datos solicitados y presiona ok	4. El sistema genera un archivo temporal en formato rib el cual es posteriormente pasado al programa rendrib, en caso que se seleccione la opción render en red se pasa al programa farm (script en perl que usa rsh para llamar rendrib en máquinas remotas) el cual se encarga de generar archivos de imagen .tiff en el intervalo de cuadros especificado, si se eligió la opción crear vídeo estas imágenes son transformadas a ppm y posteriormente a MPEG-1 y posteriormente eliminadas.
	5. Si se eligió grabar el resultado el sistema usa el caso de uso desplegar último rendering para mostrar el resultado.

Curso alterno de los eventos

3.a El usuario presiona cancelar (cancel)
4.a El sistema cierra el dialogo y no realiza ninguna acción.
5.a Si dentro de las opciones de render no se escogió la opción grabar el sistema despliega de inmediato una ventana que muestra el render.

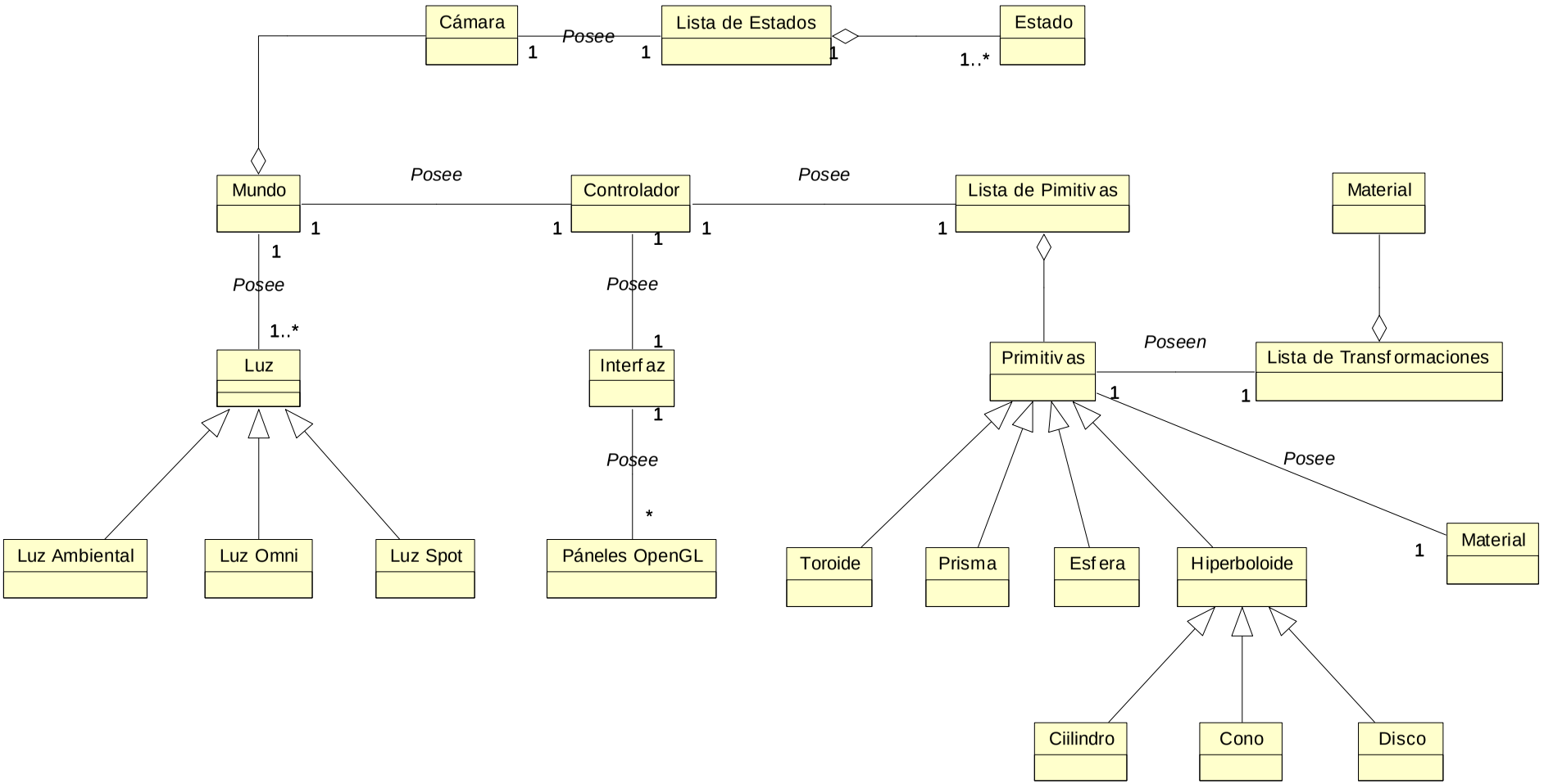
Nombre	Desplegar el último rendering.
Actores	Usuario
Resumen	Este caso de uso le permite a los usuarios del sistema ver el último render realizado si el usuario escogió durante las opciones del render la opción grabar.
Tipo	Primario y esencial
Propósito	Permitir al usuario la visualización del último rendering.
Referencias cruzadas	Requerimientos funcionales: 29 Casos de uso: Renderizar Escena (<<uses>>), Generar imagen o vídeo final.

Curso normal de los eventos	
Acciones de los actores	Respuesta del sistema
1. El usuario presiona el botón ver último render (view last rendering)	2. El sistema verifica en las opciones de generación del render se haya especificado grabar el archivo y obtiene el nombre con el cual se grabó la salida.
	4. El sistema despliega una ventana con la imagen o secuencia de imágenes usando el programa iv para esto. Si es un vídeo se hace por medio del programa plaympeg.

Curso alterno de los eventos

4.a Si en la verificación del paso 2 no se encuentran activas las opciones de grabar entonces el programa despliega un dialogo diciendo que no existe archivo previo de renderización.

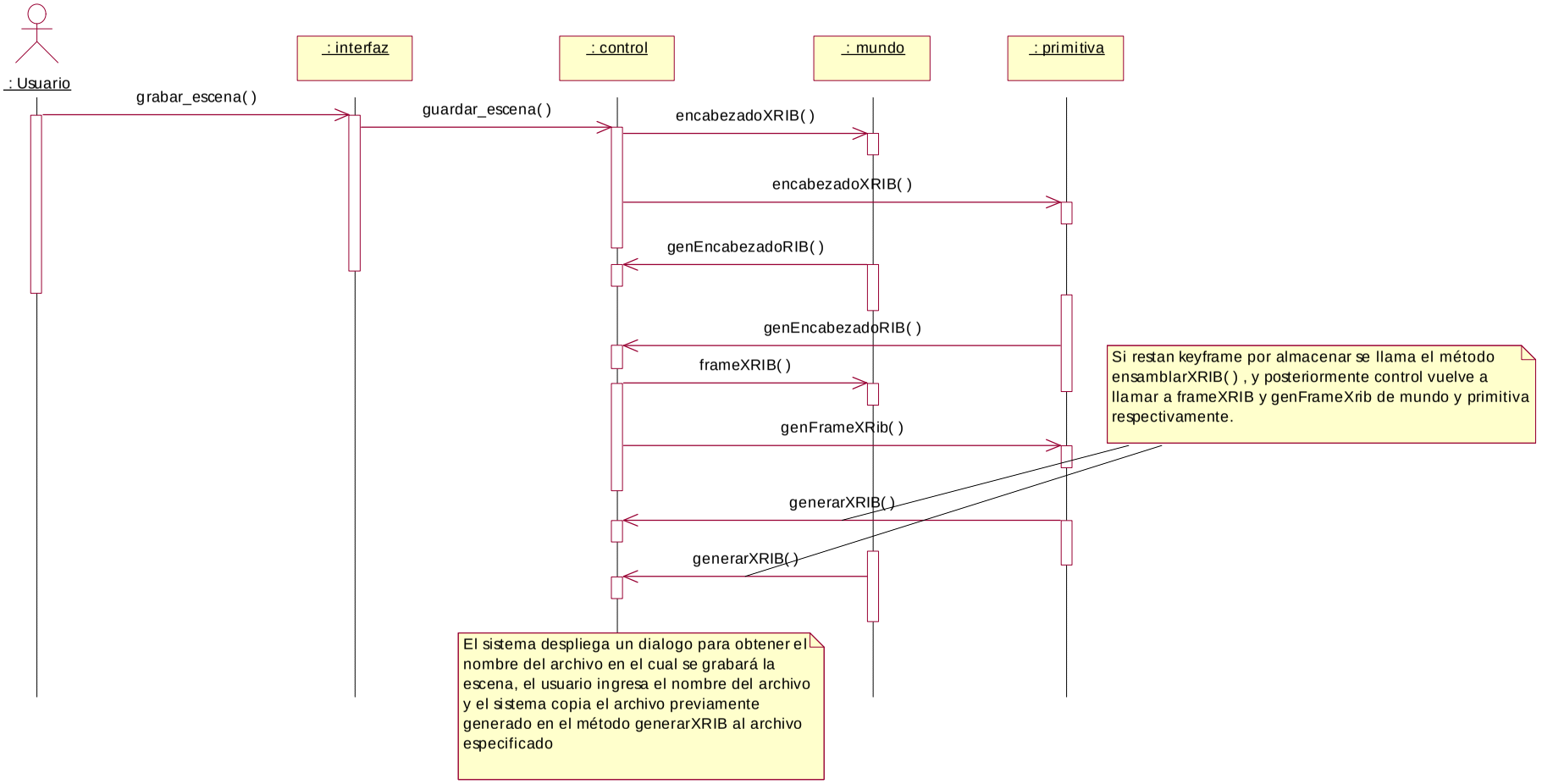
5.3. MODELO CONCEPTUAL



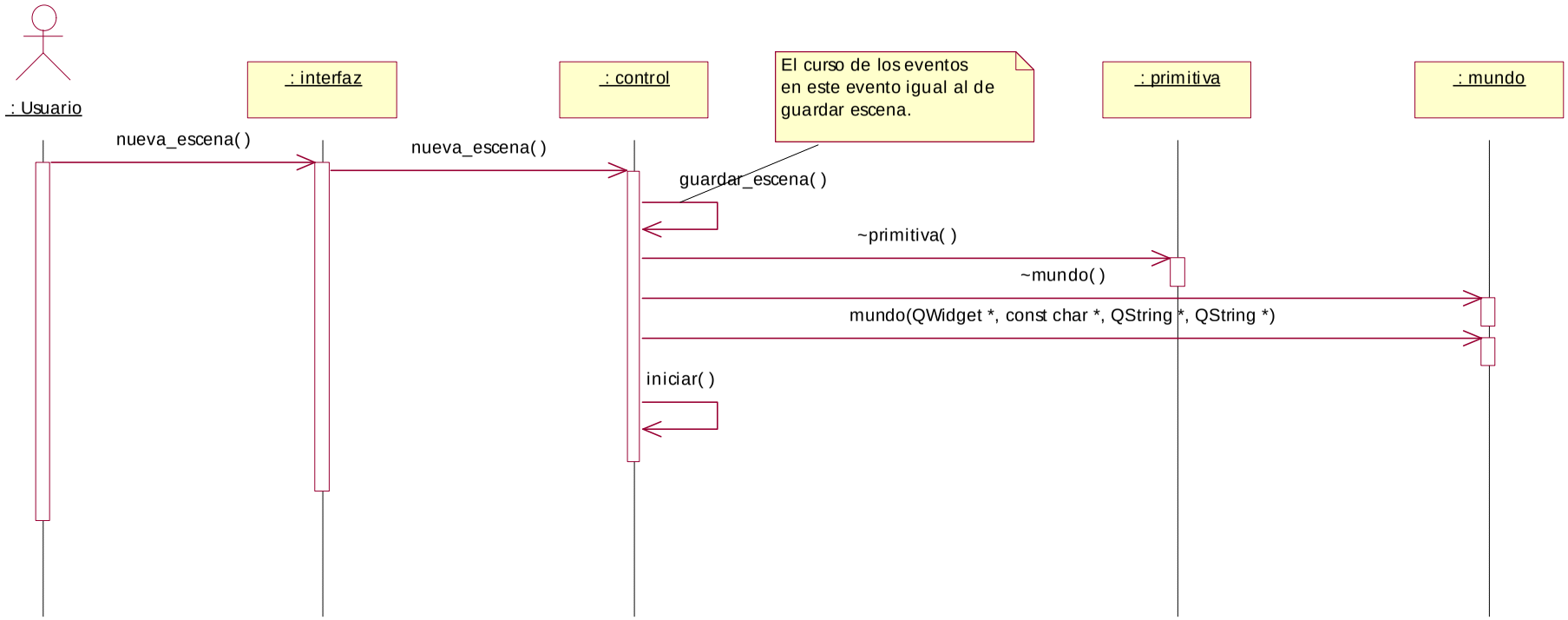
6. Diseño del Sistema

6.1 Diagramas de Secuencia

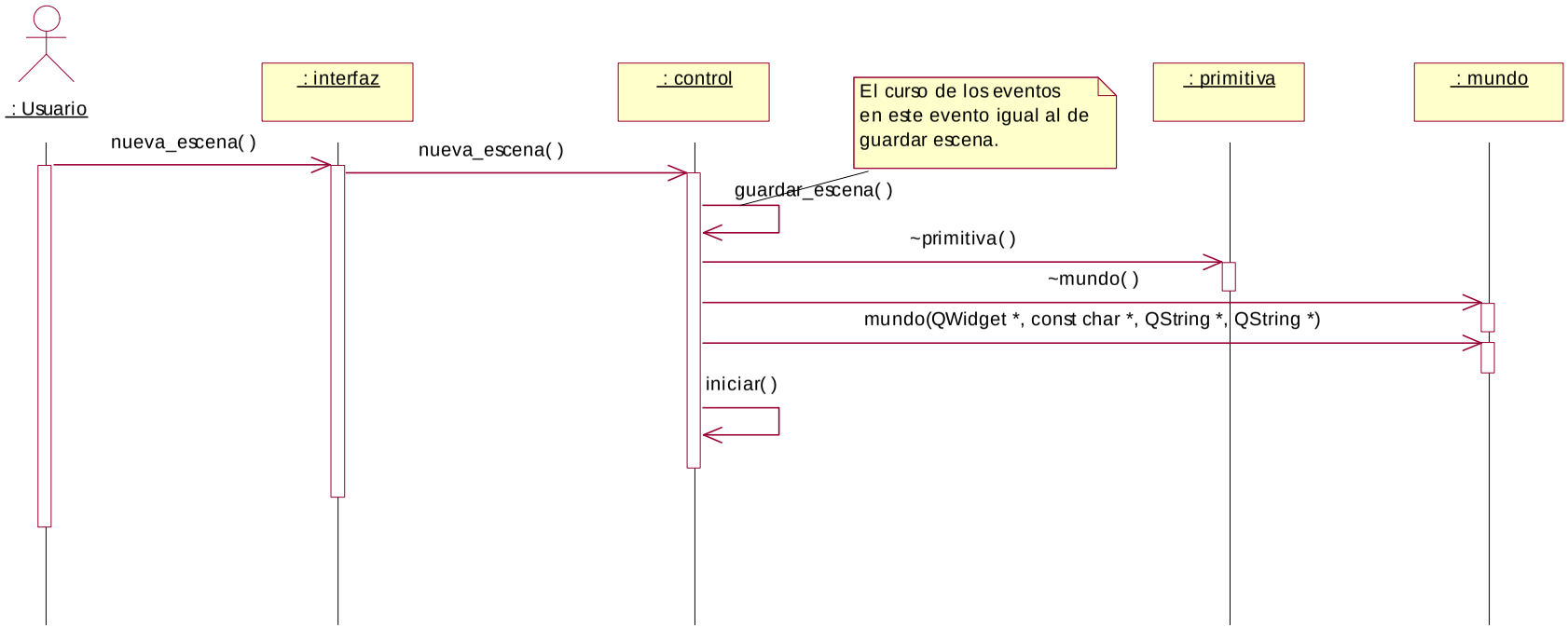
Caso de Uso: Grabar Escena



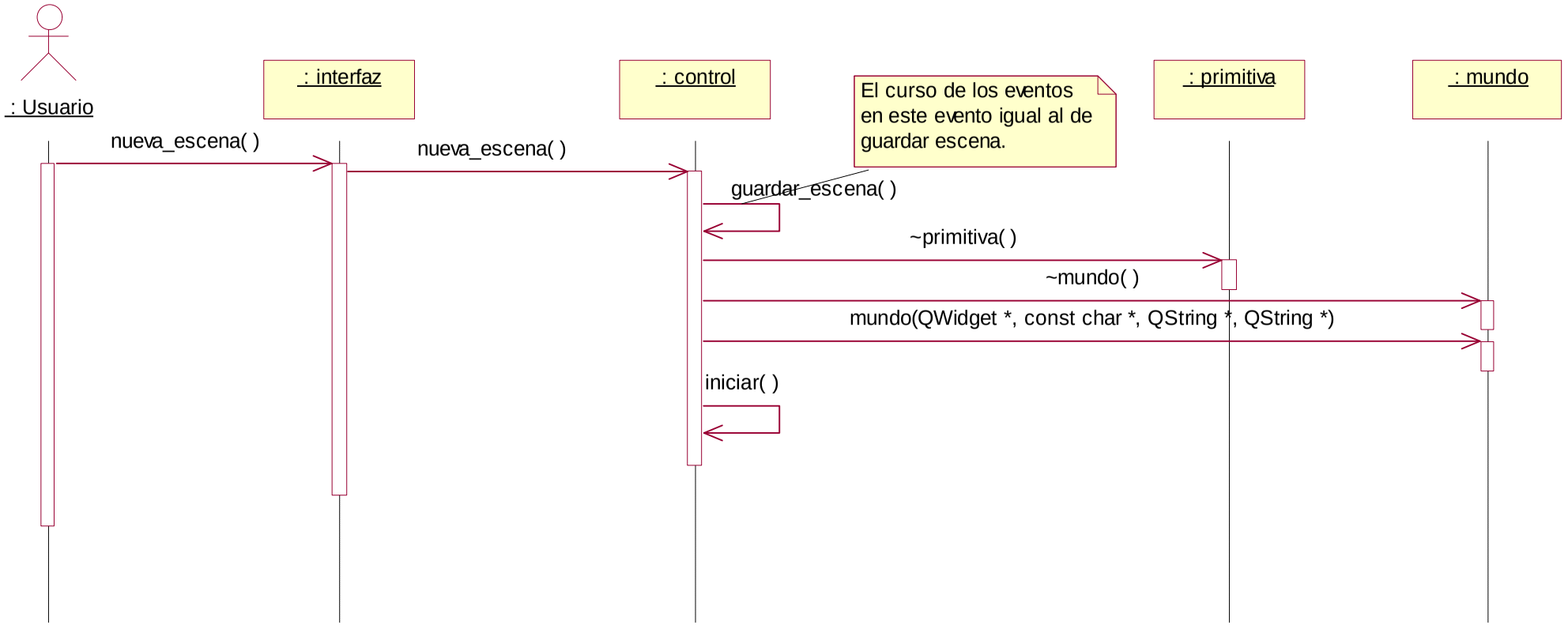
Caso de Uso: Cerrar Escena



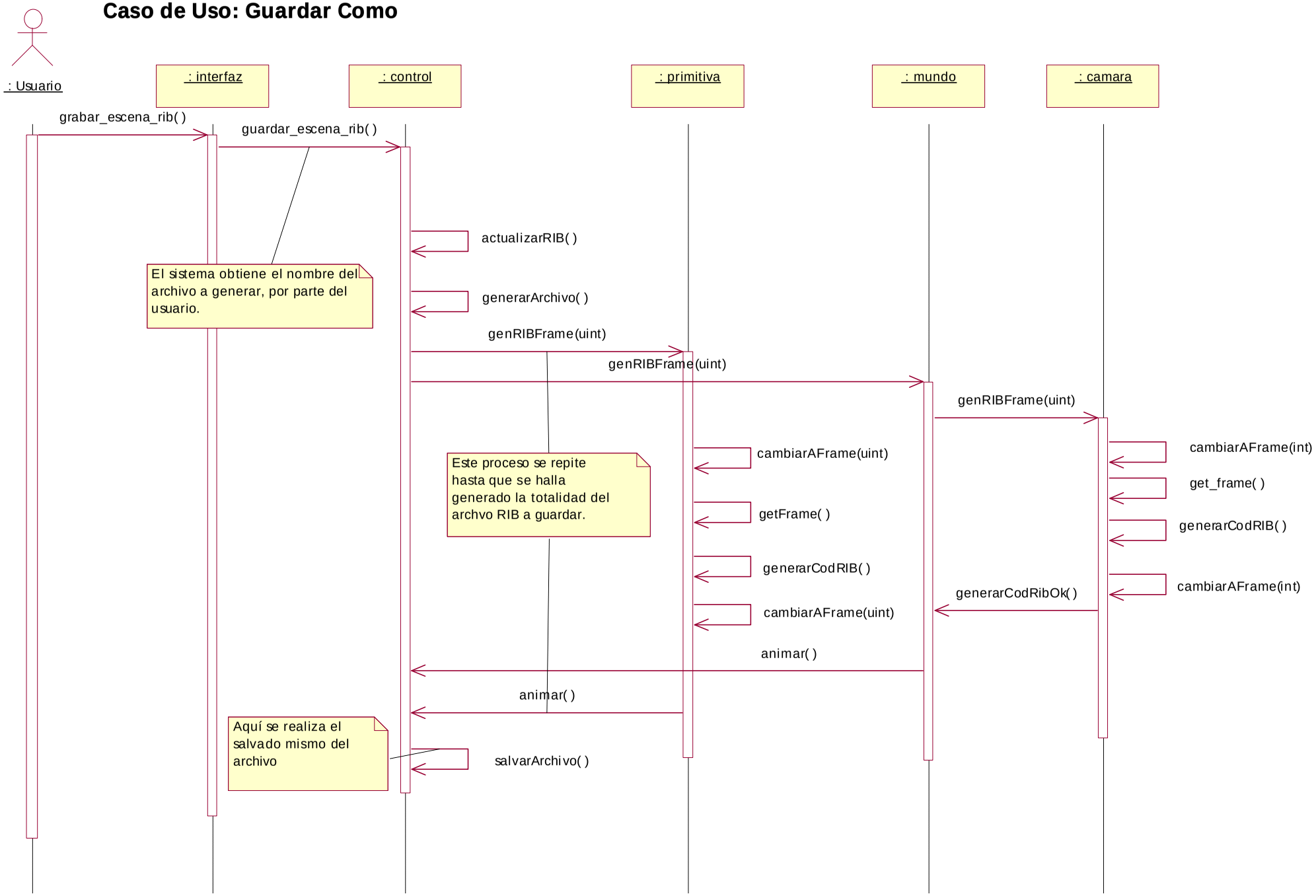
Caso de Uso: Crear Escena



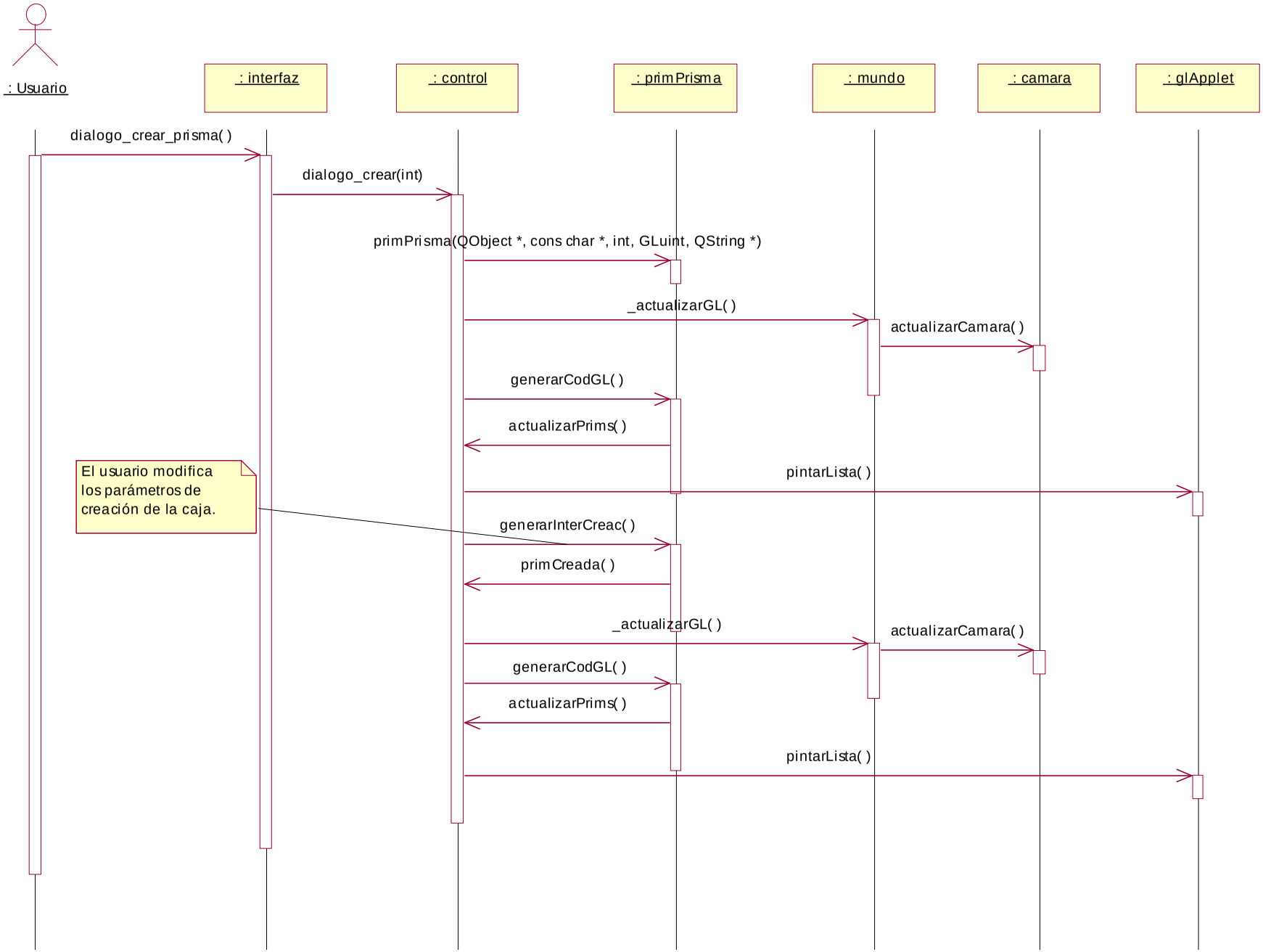
Caso de Uso: Abrir Escena



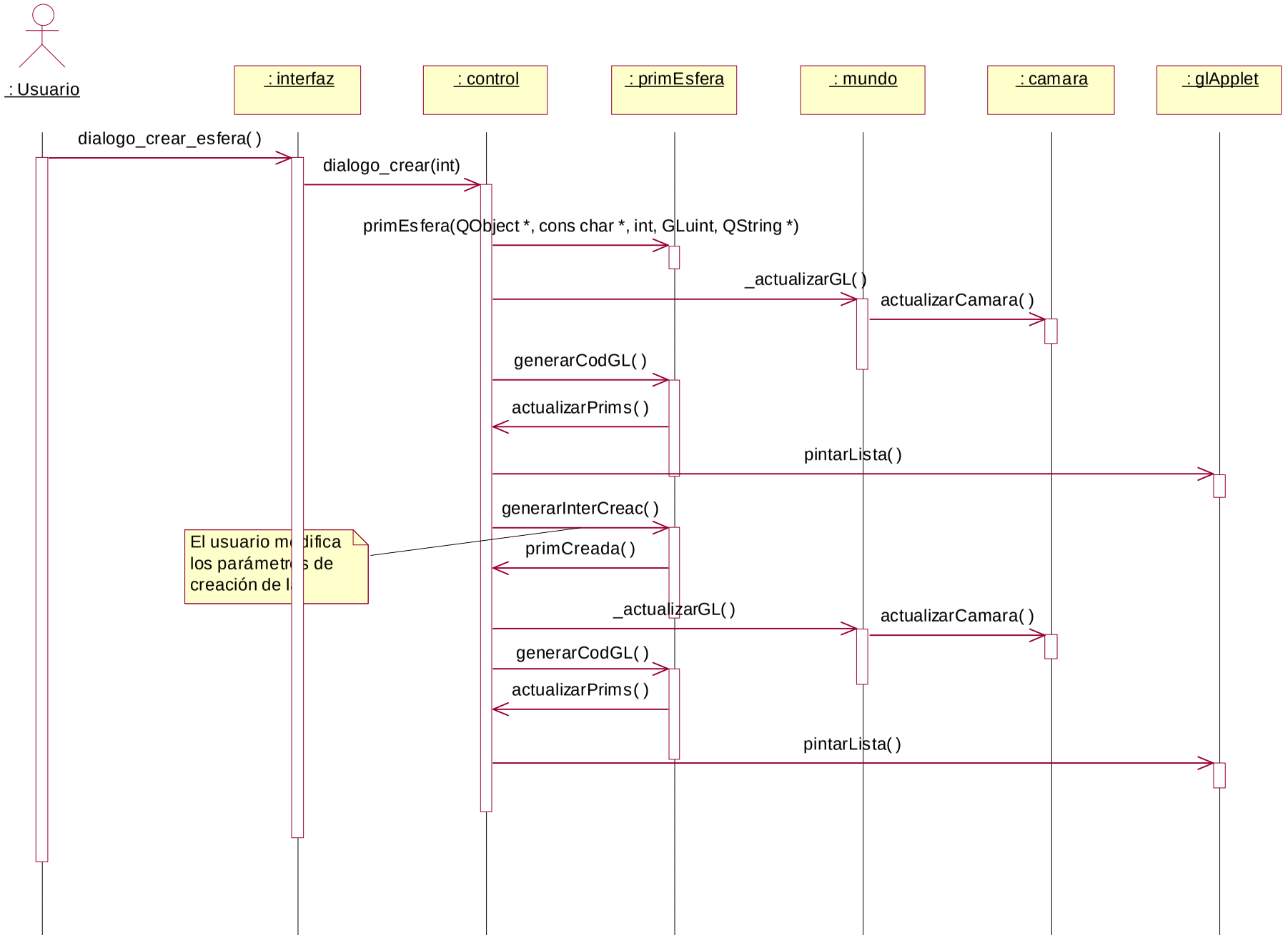
Caso de Uso: Guardar Como



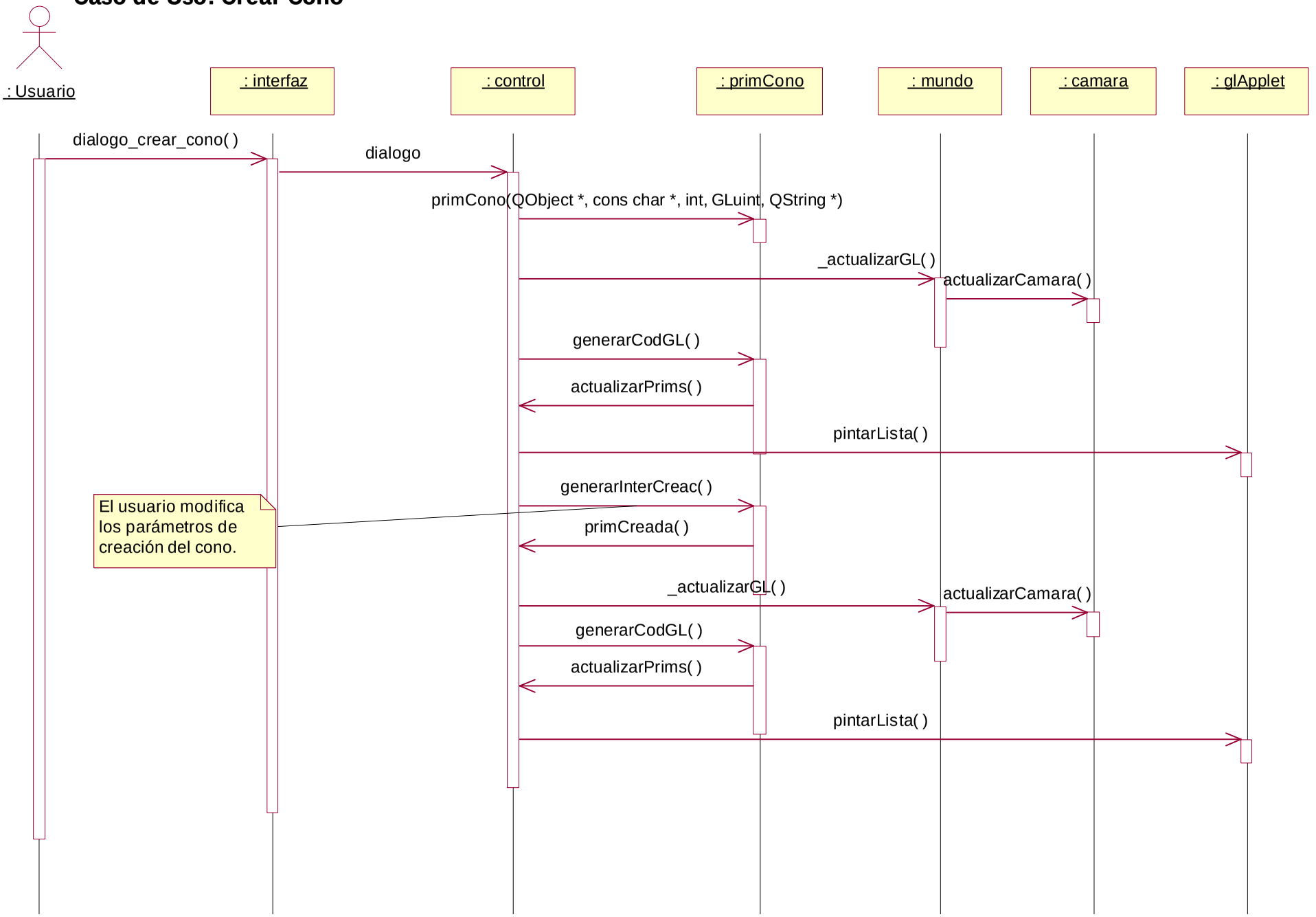
Caso de Uso: Crear Caja



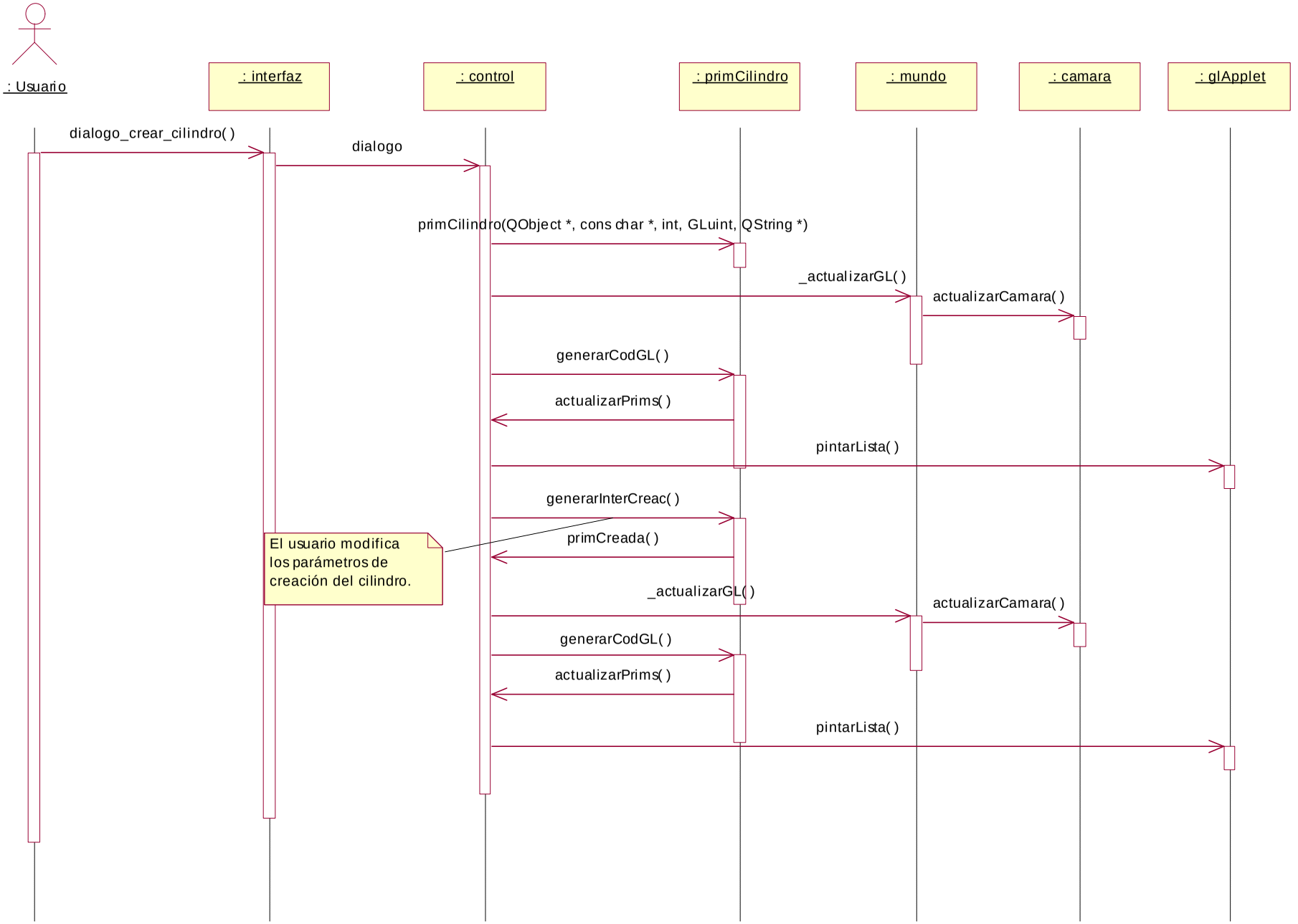
Caso de Uso: Crear Esfera



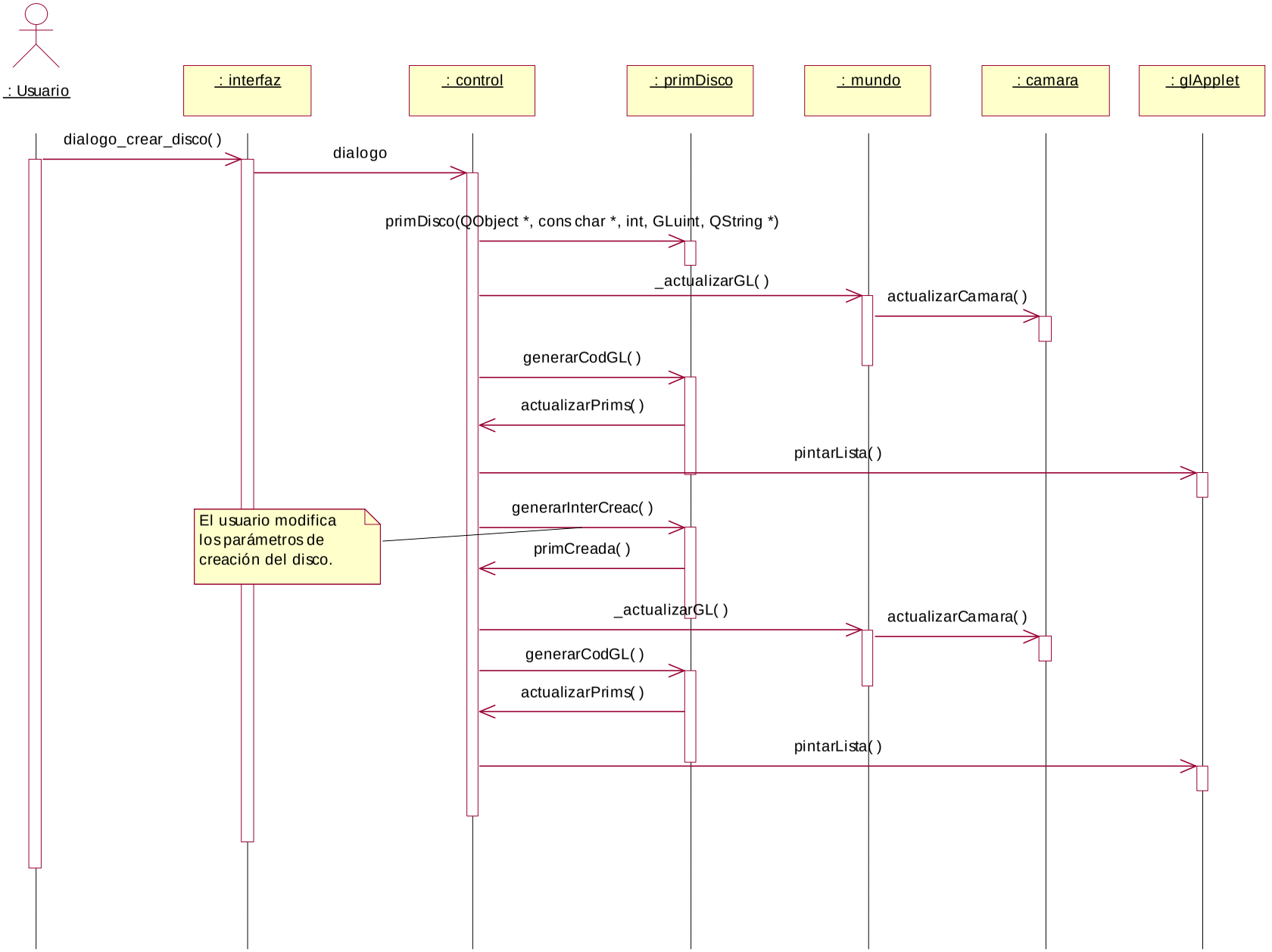
Caso de Uso: Crear Cono



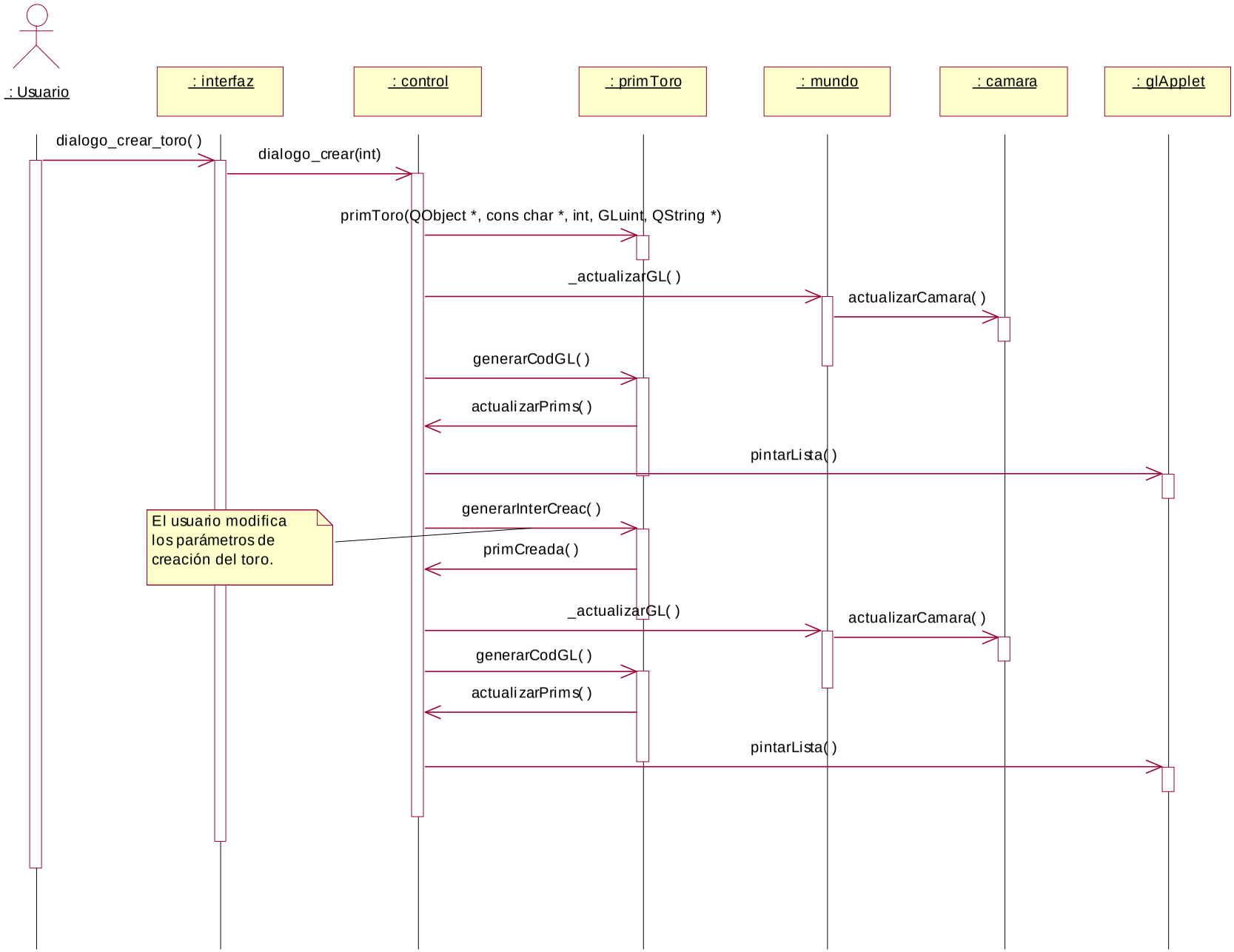
Caso de Uso: Crear Cilindro



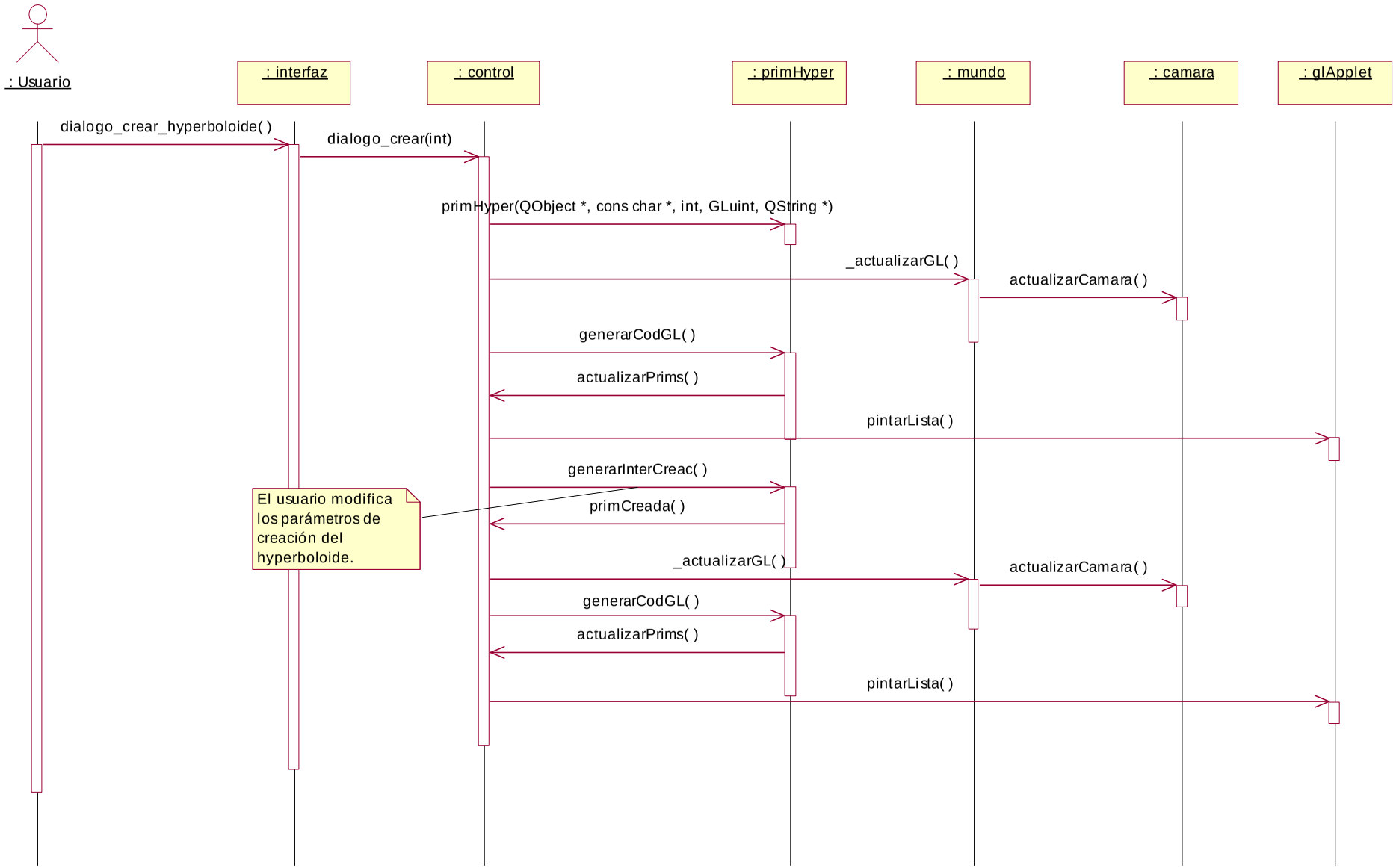
Caso de Uso: Crear Disco



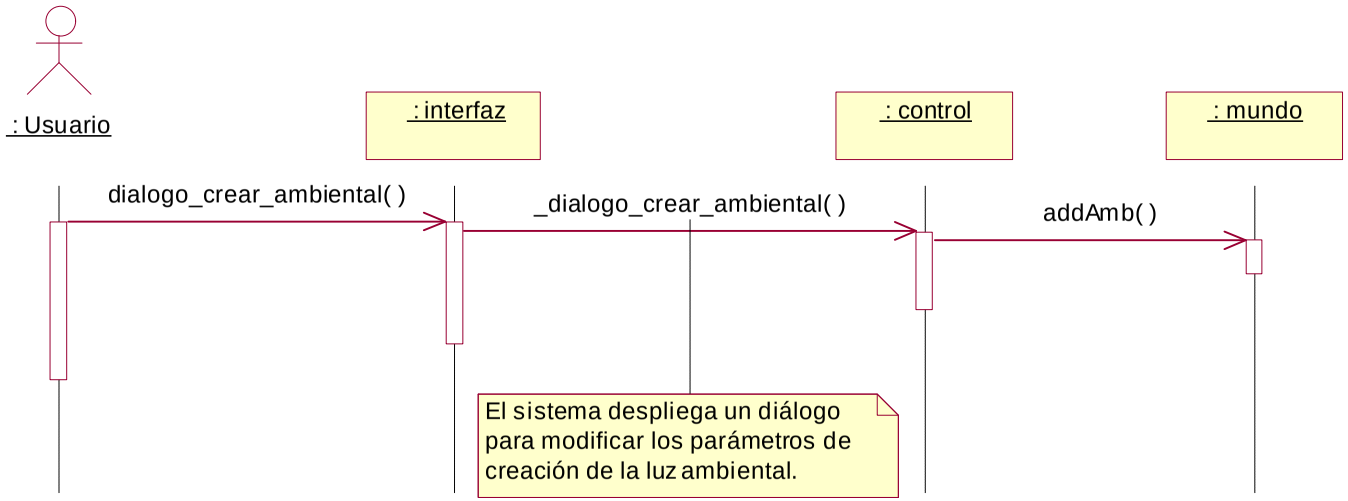
Caso de Uso: Crear Toroide



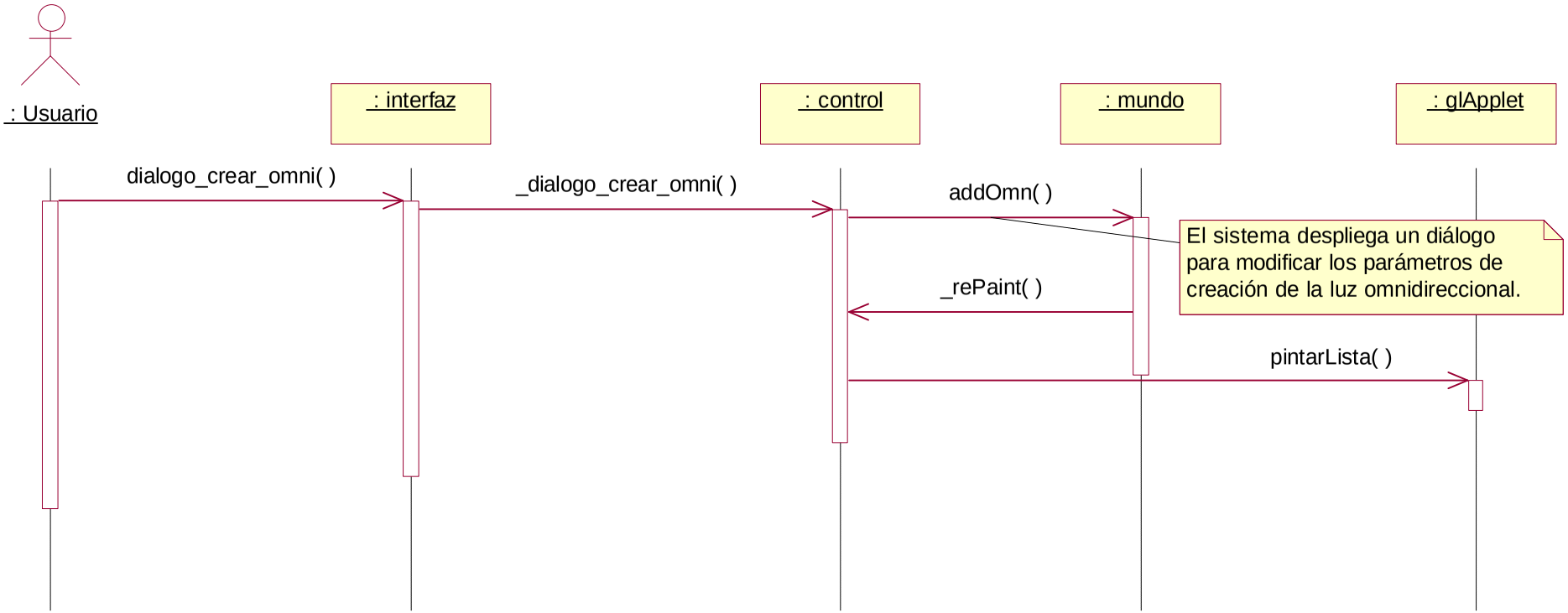
Caso de Uso: Crear Hiperboloide



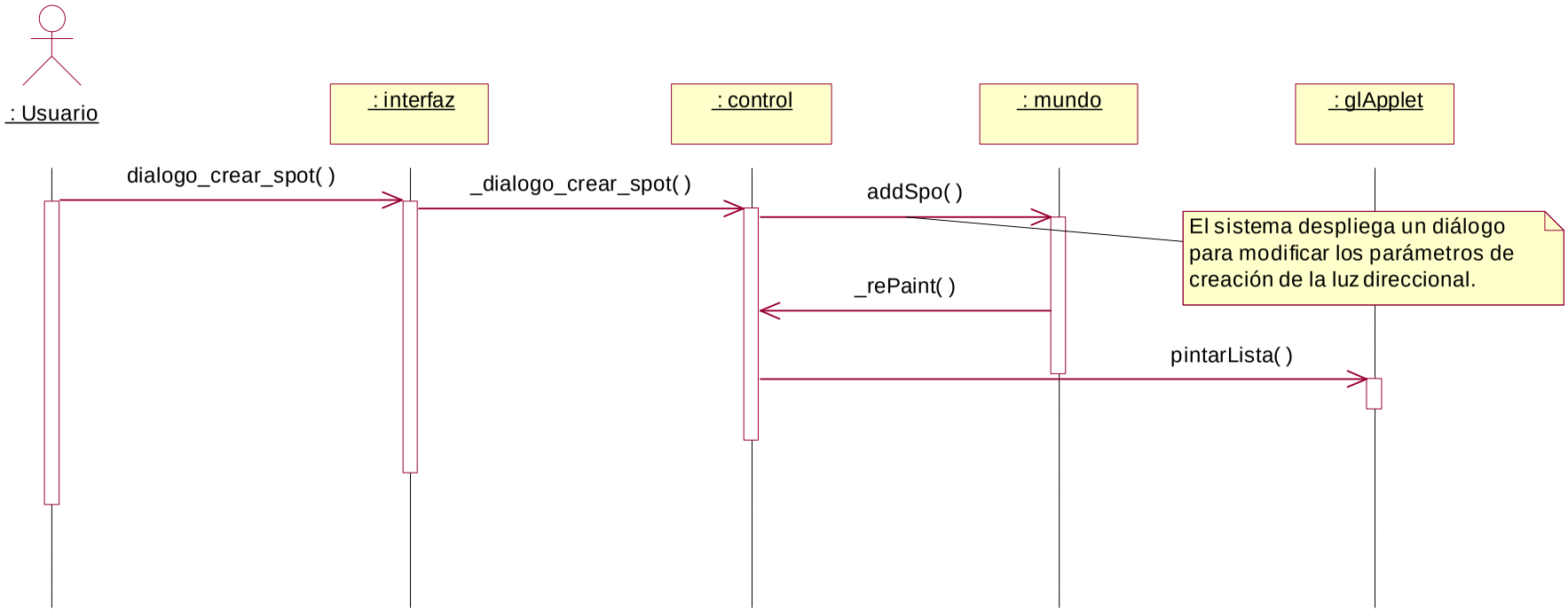
Caso de Uso: Crear Luz Ambiental



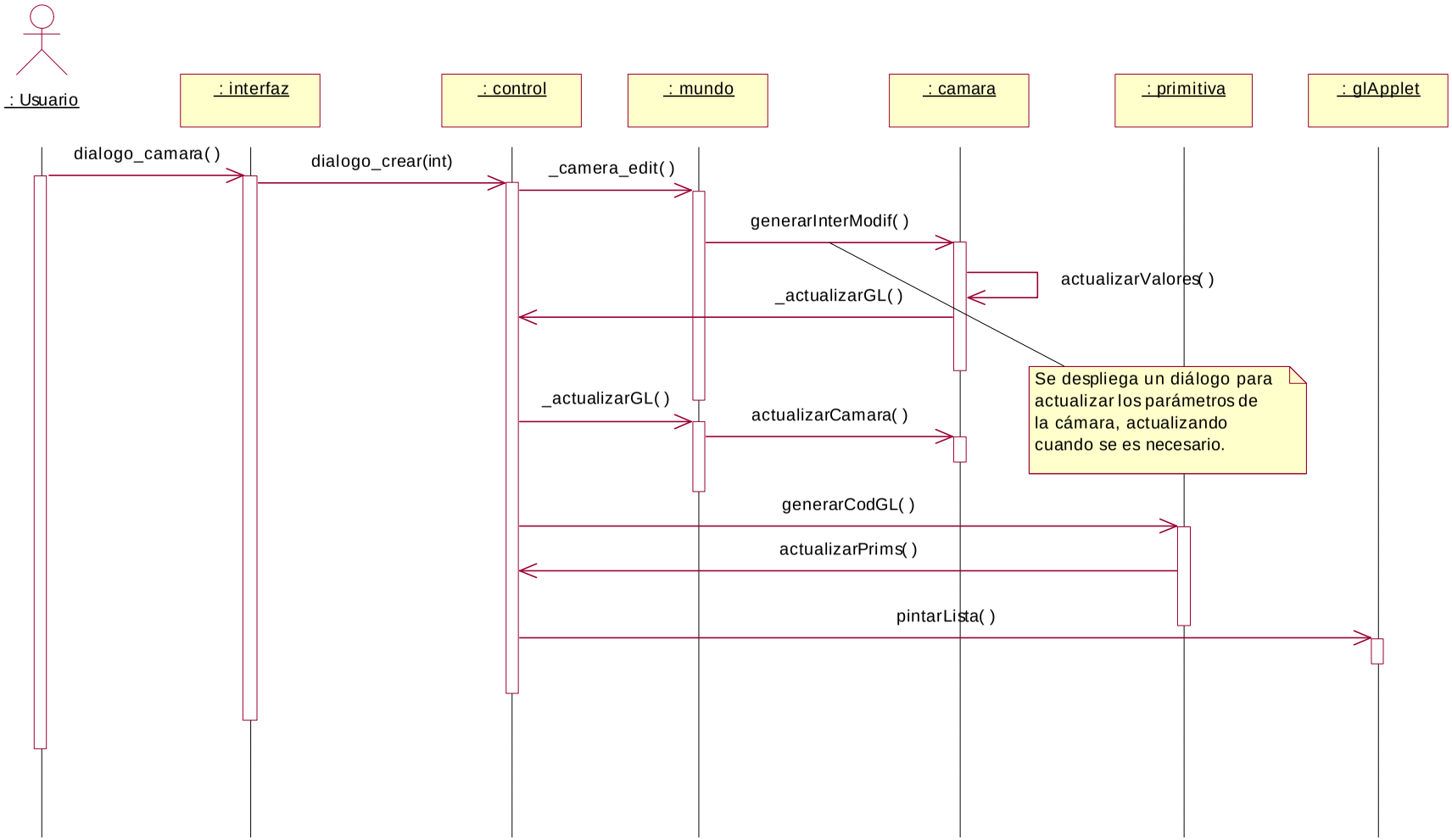
Caso de Uso: Crear Luz Omnidireccional



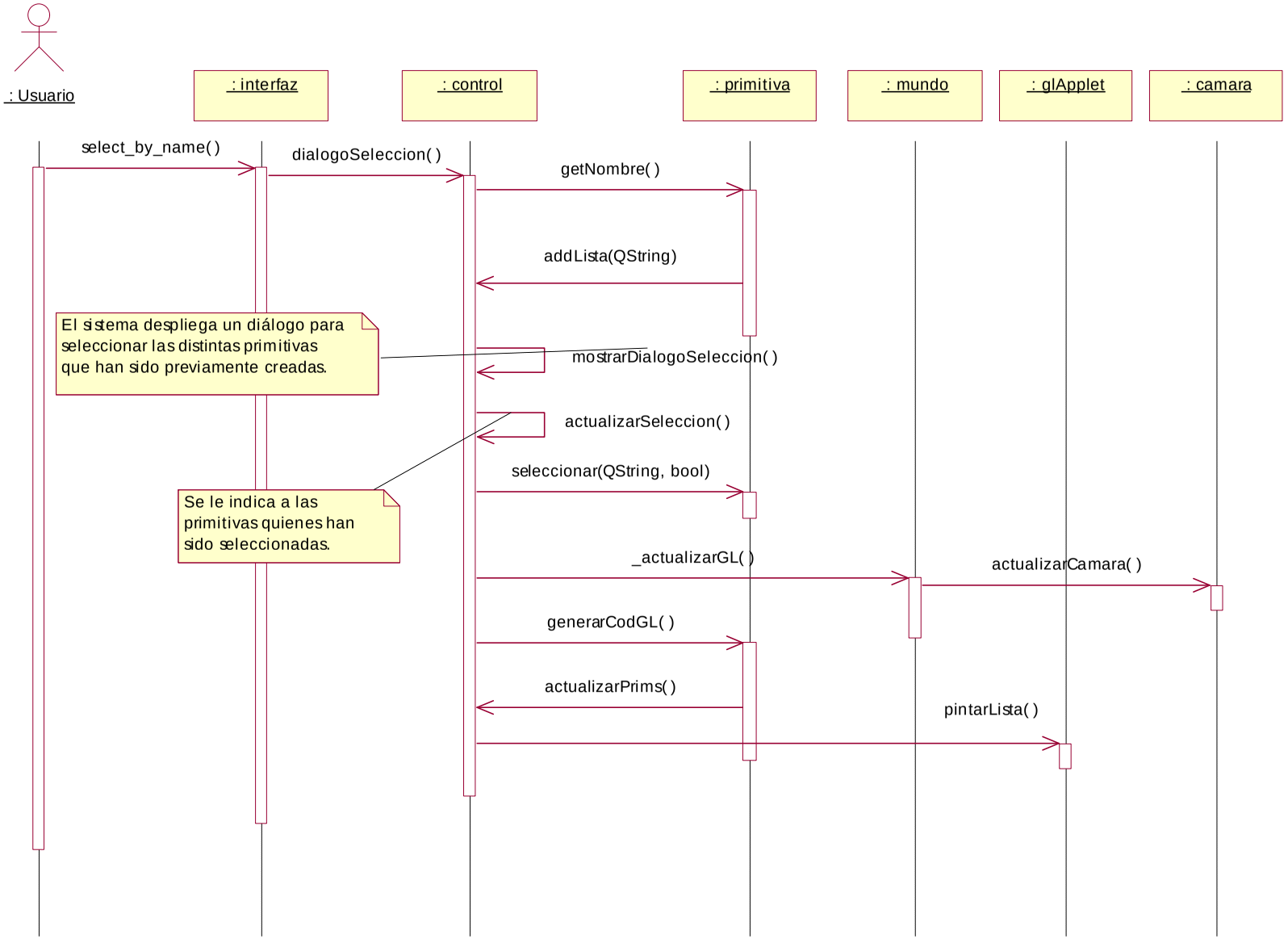
Caso de Uso: Crear Luz Direccional tipo Reflector



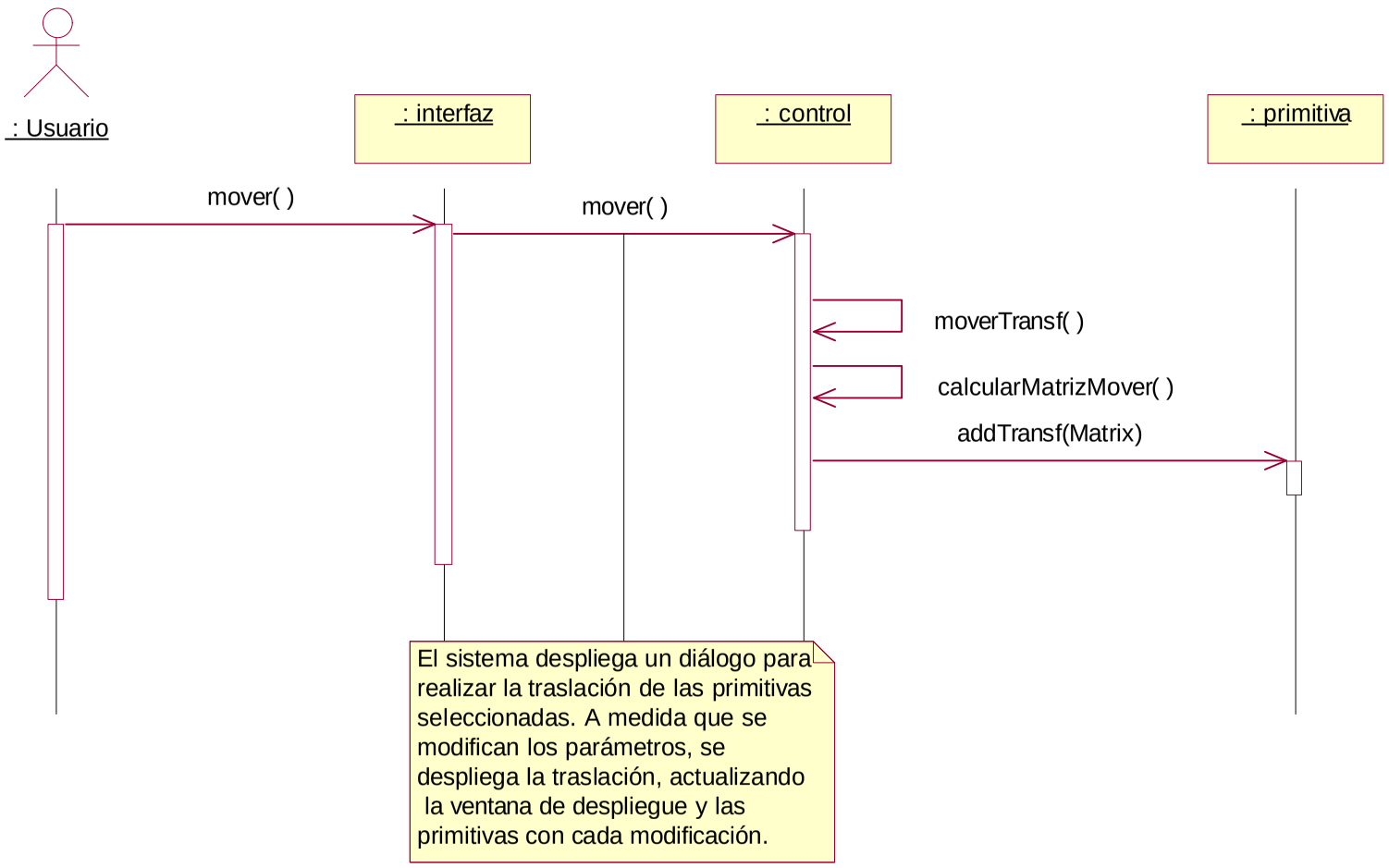
Caso de Uso: Posicionar Cámara



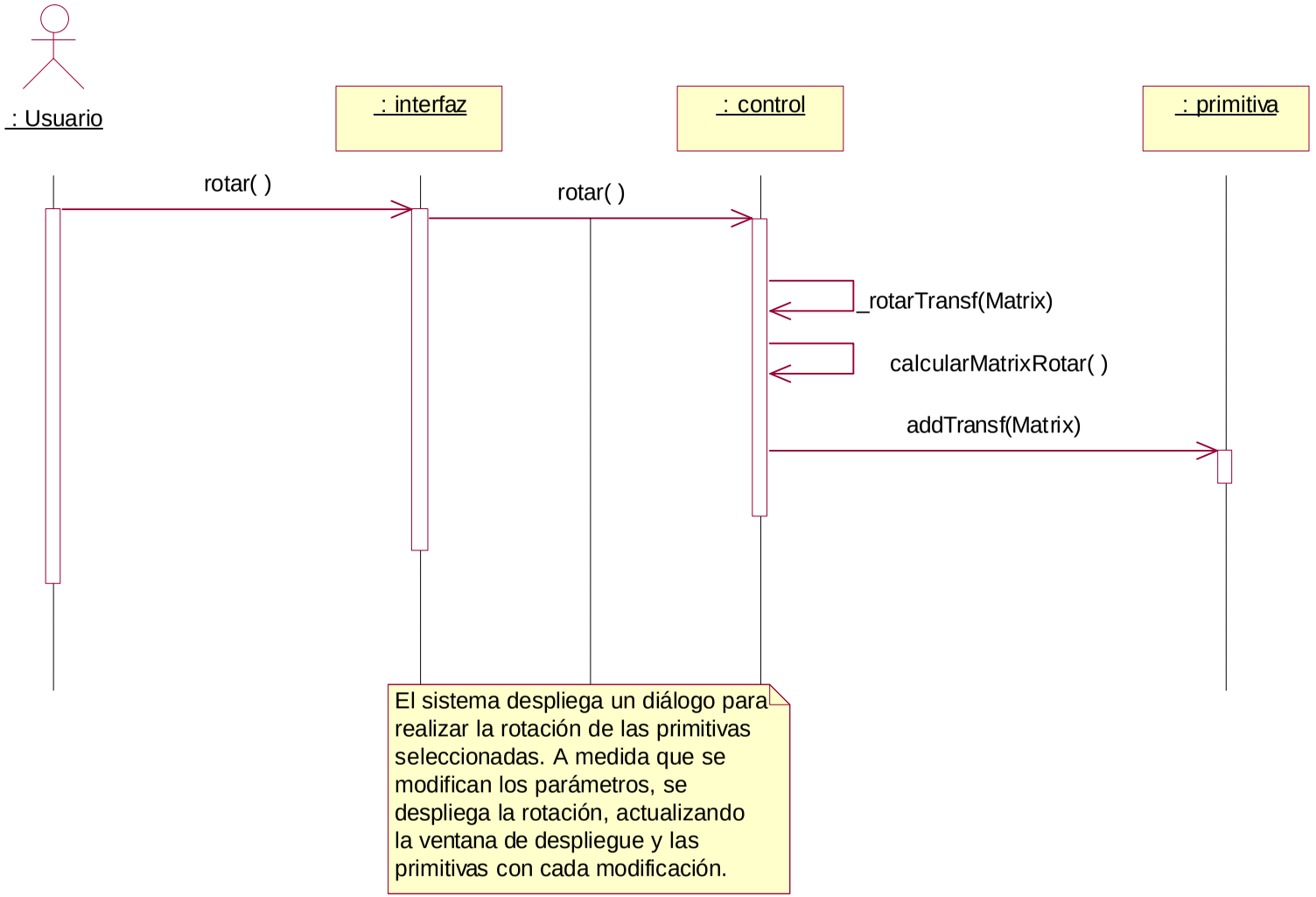
Caso de Uso: Seleccionar Objetos



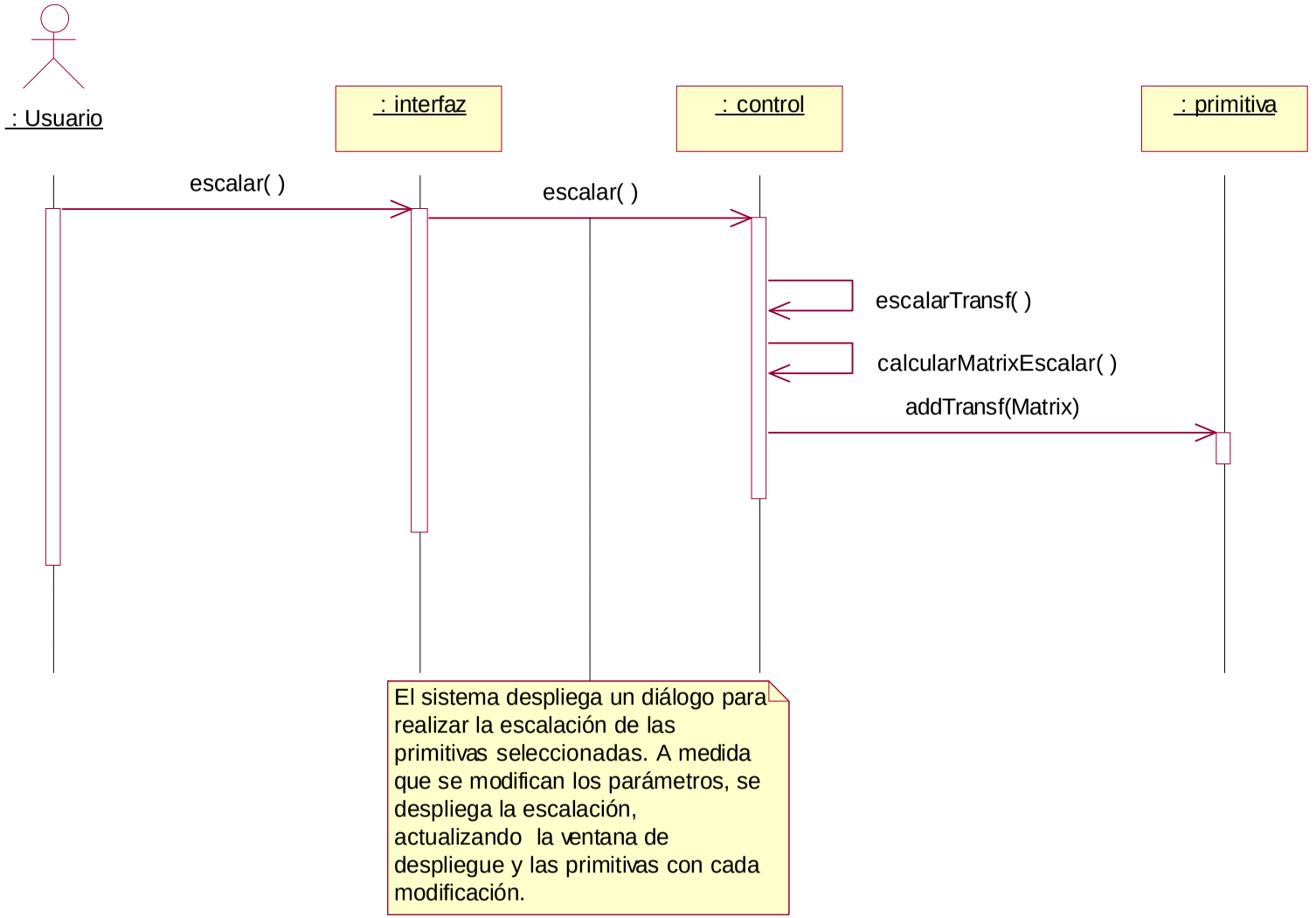
Caso de Uso: Trasladar Objetos



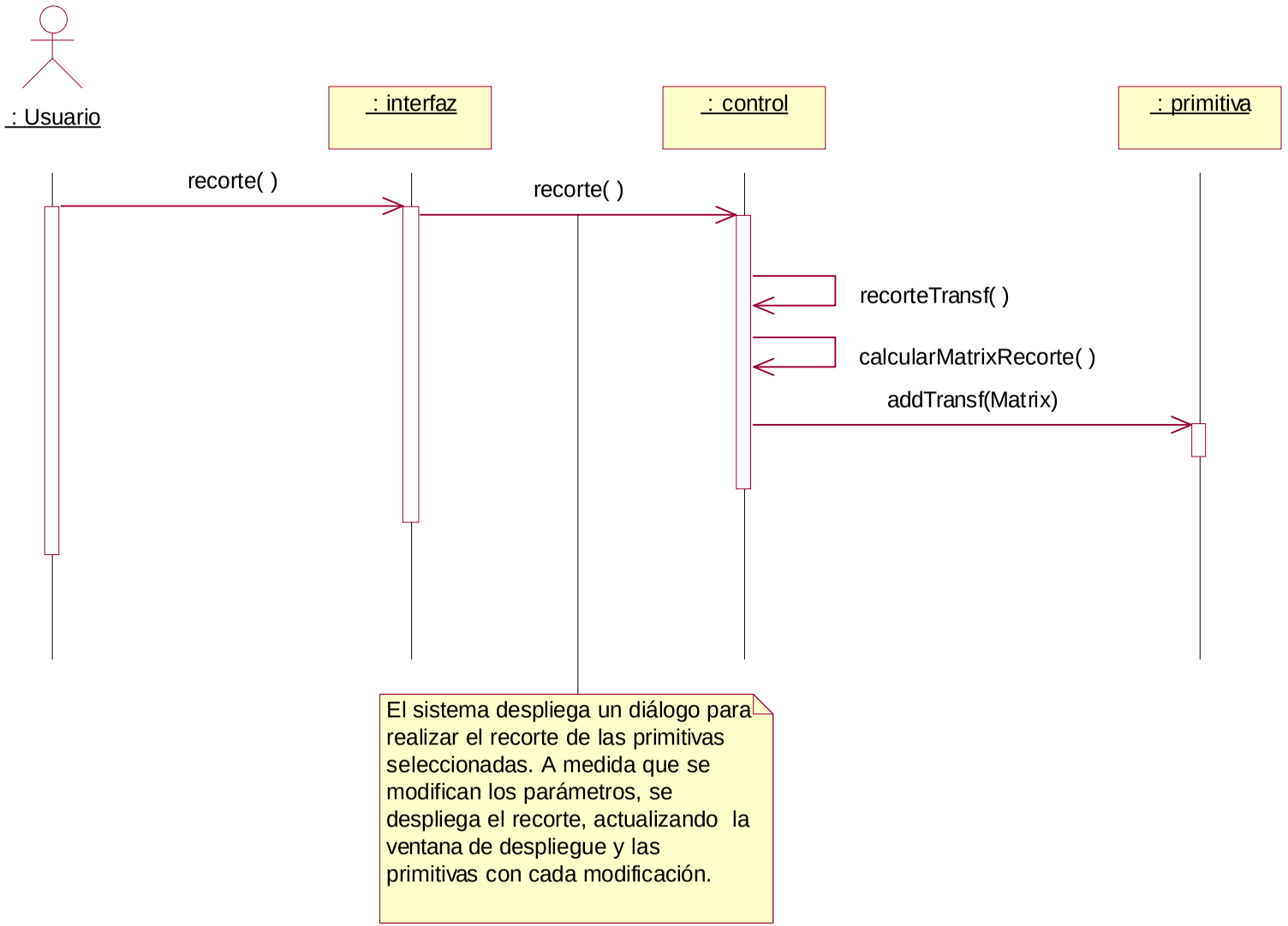
Caso de Uso: Rotar Objetos



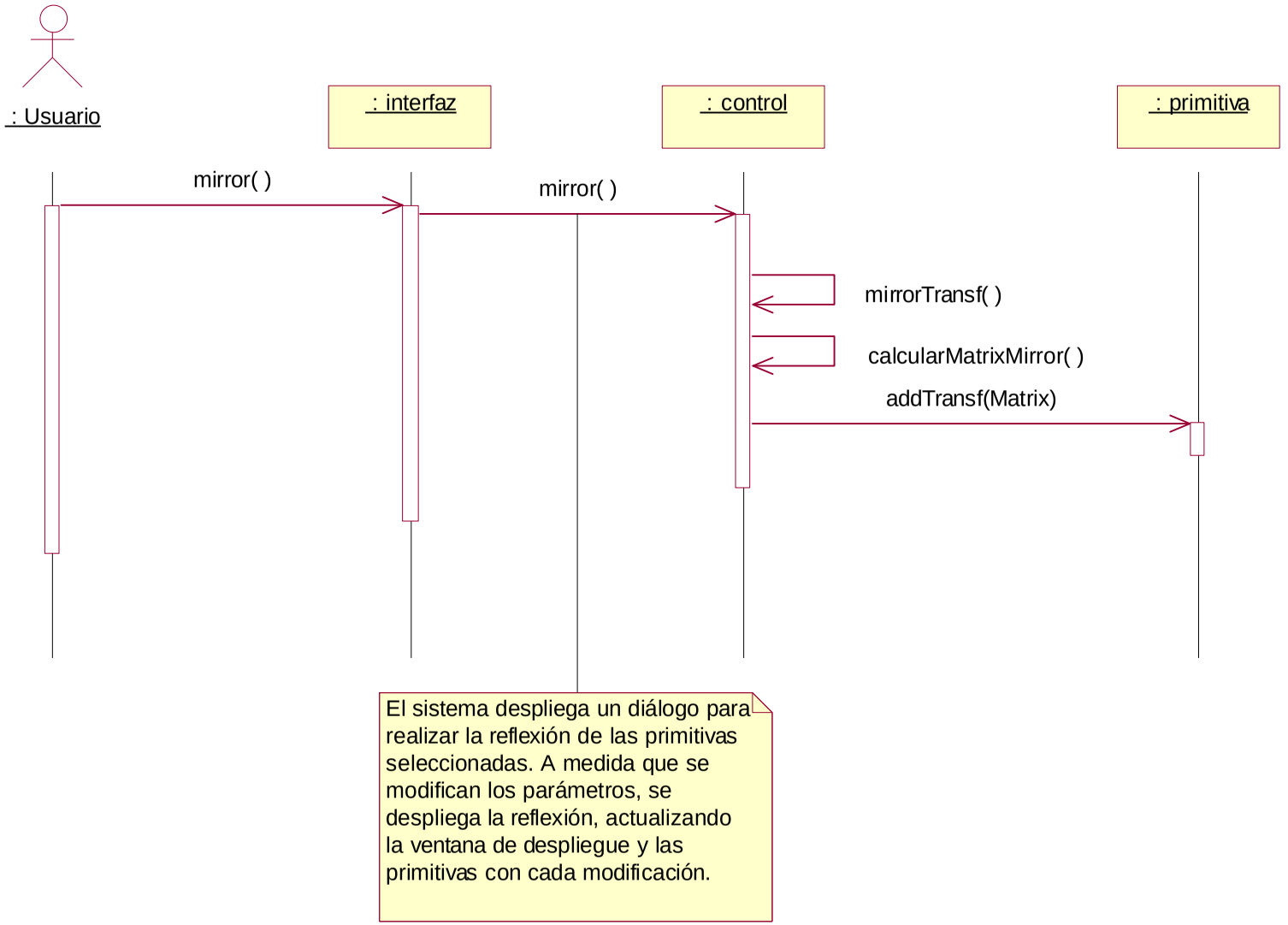
Caso de Uso: Escalar Objetos



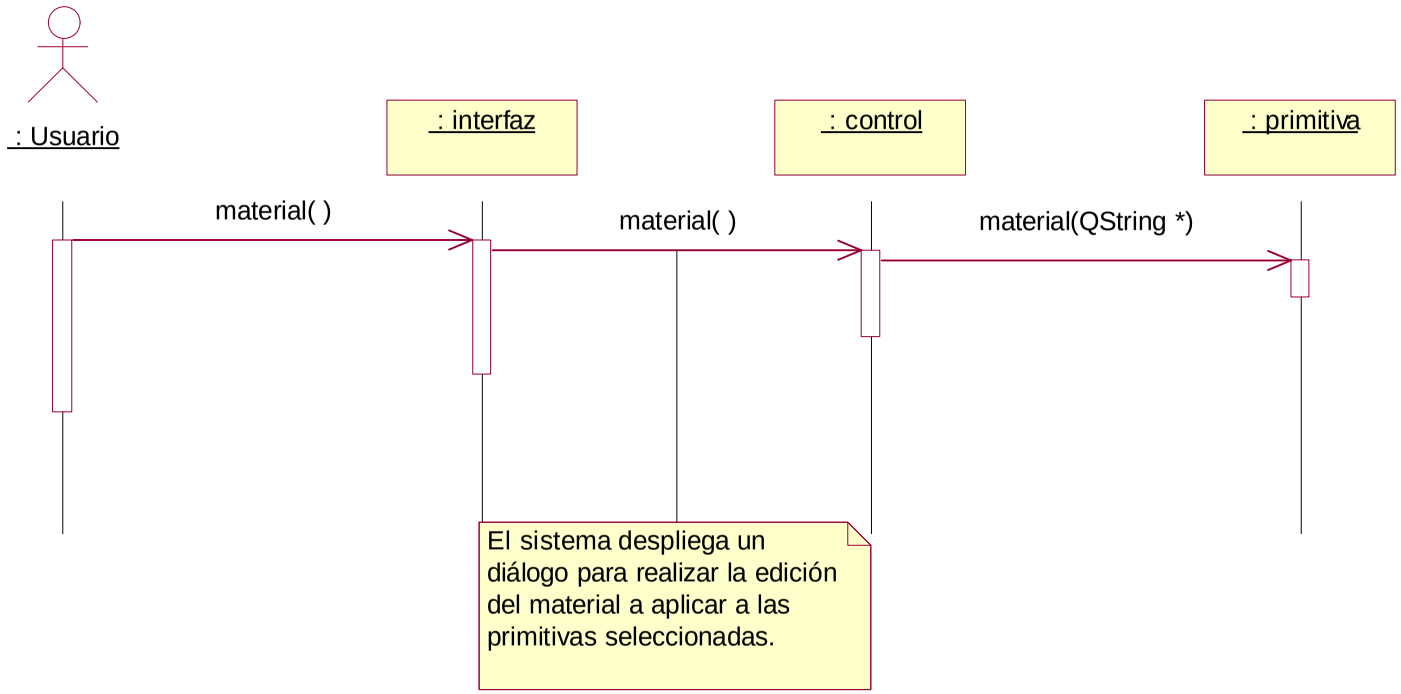
Caso de Uso: Cortar Objetos



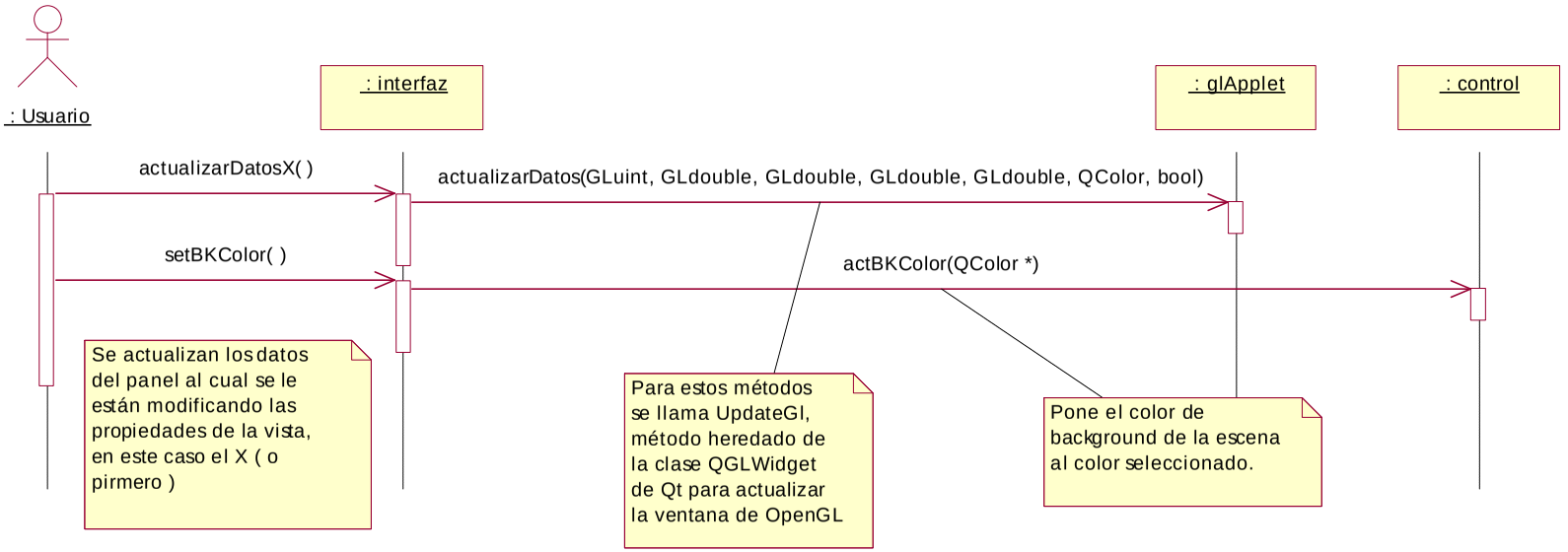
Caso de Uso: Reflejar Objetos



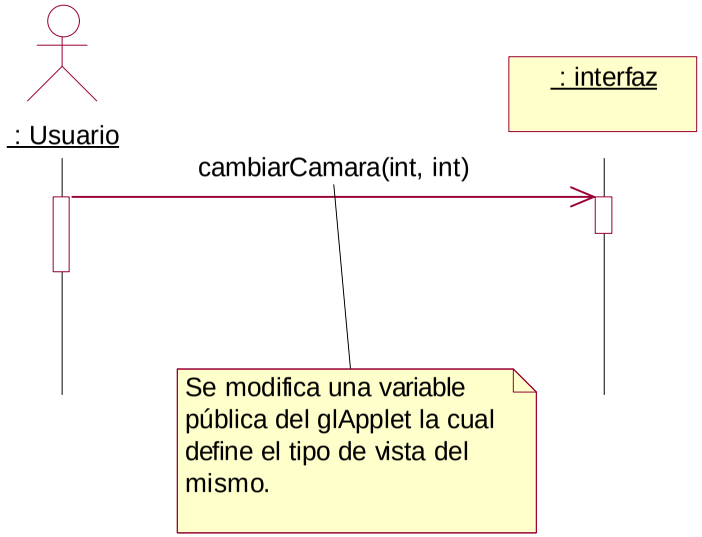
Caso de Uso: Aplicar Material



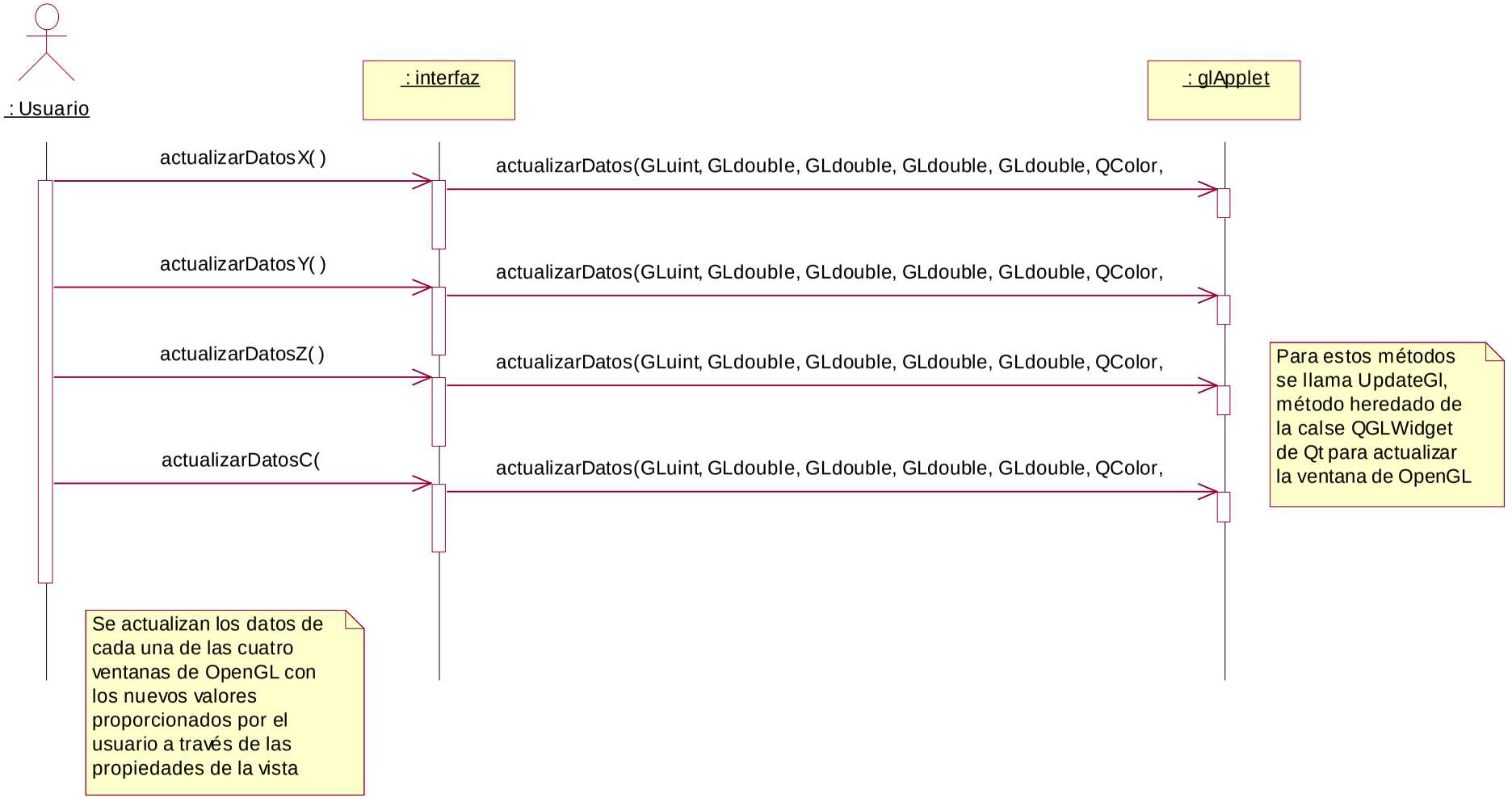
Caso de Uso: Definir propiedades del panel de visualización



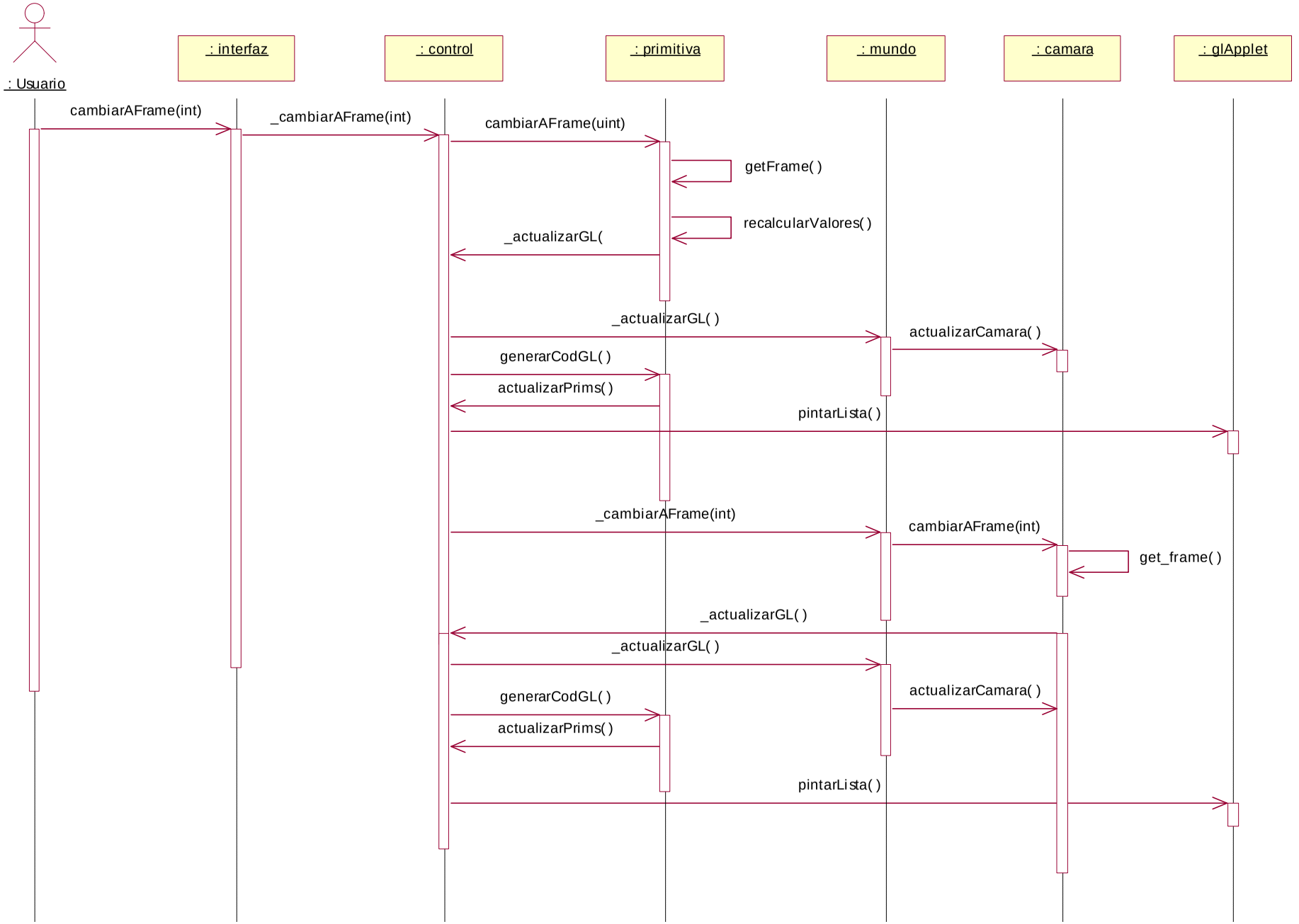
Caso de Uso: Cambiar Vistas Ortográficas



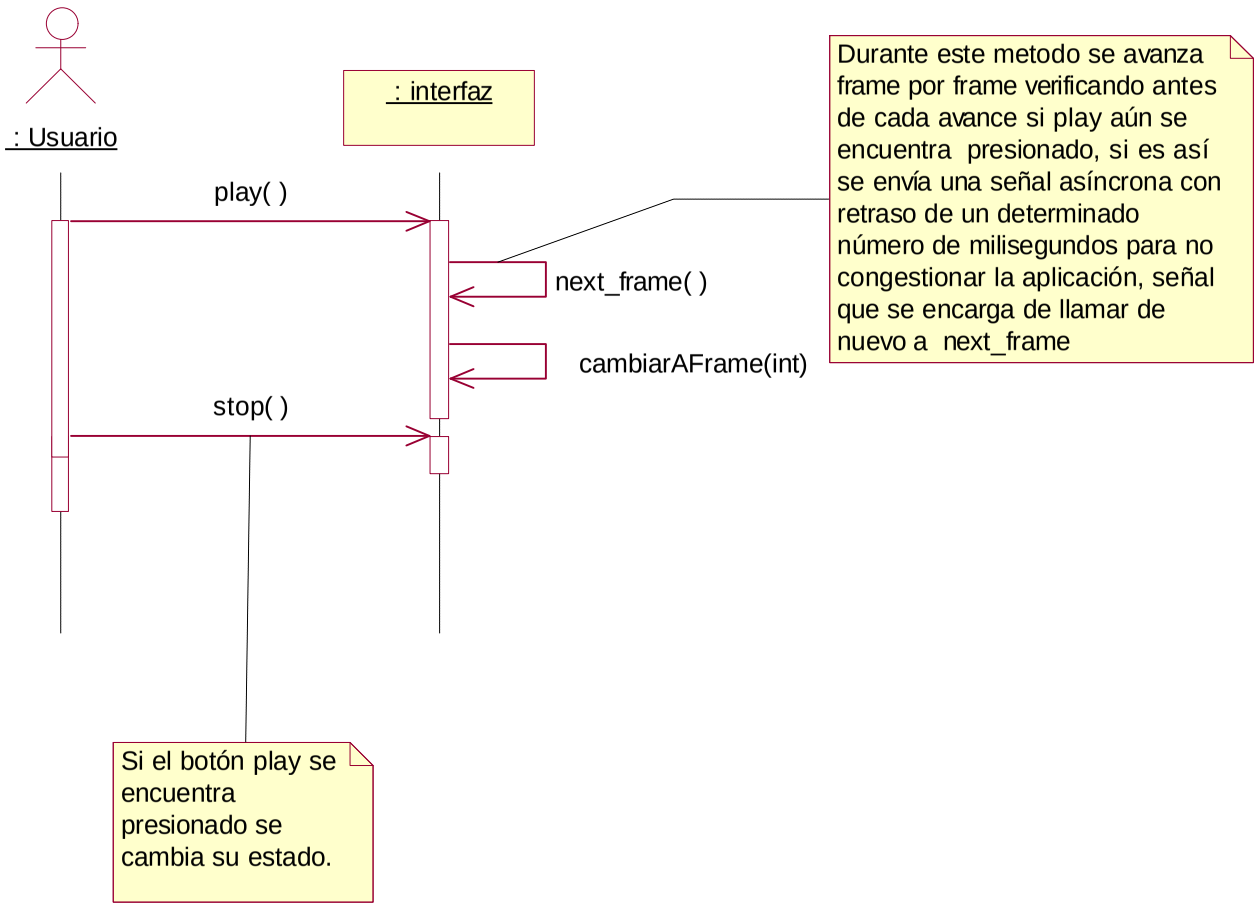
Caso de Uso: Realizar Acercamiento



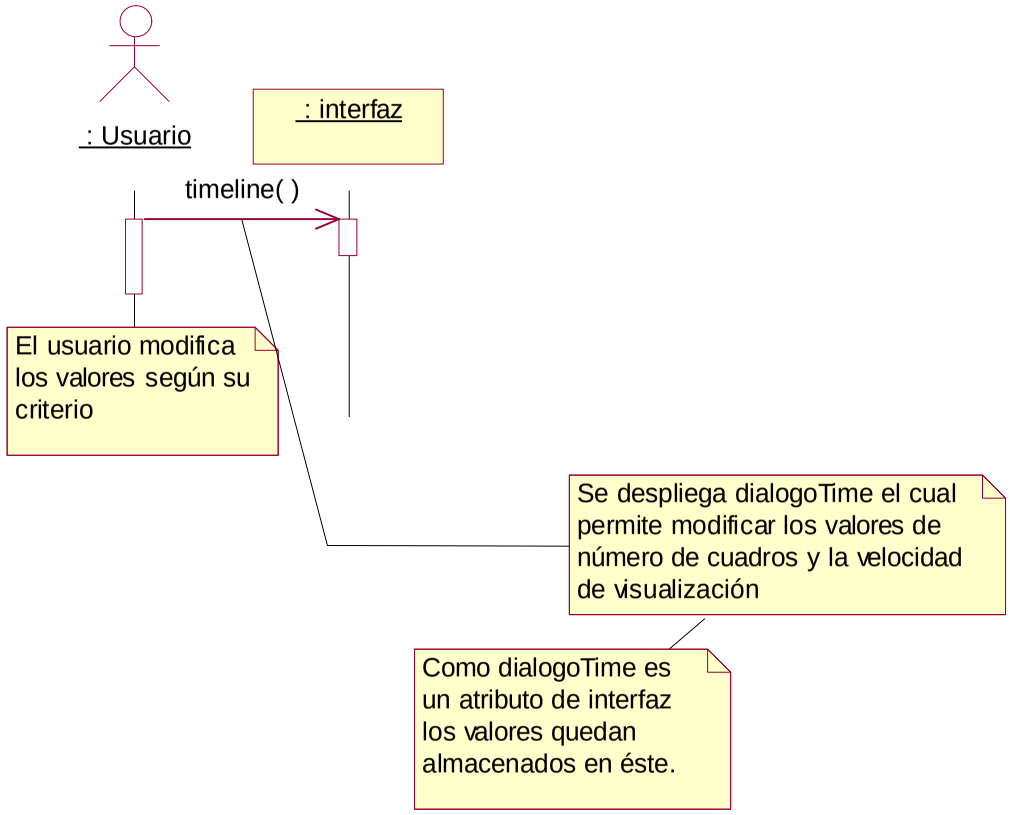
Caso de Uso: Visualizar Cuadro de la escena



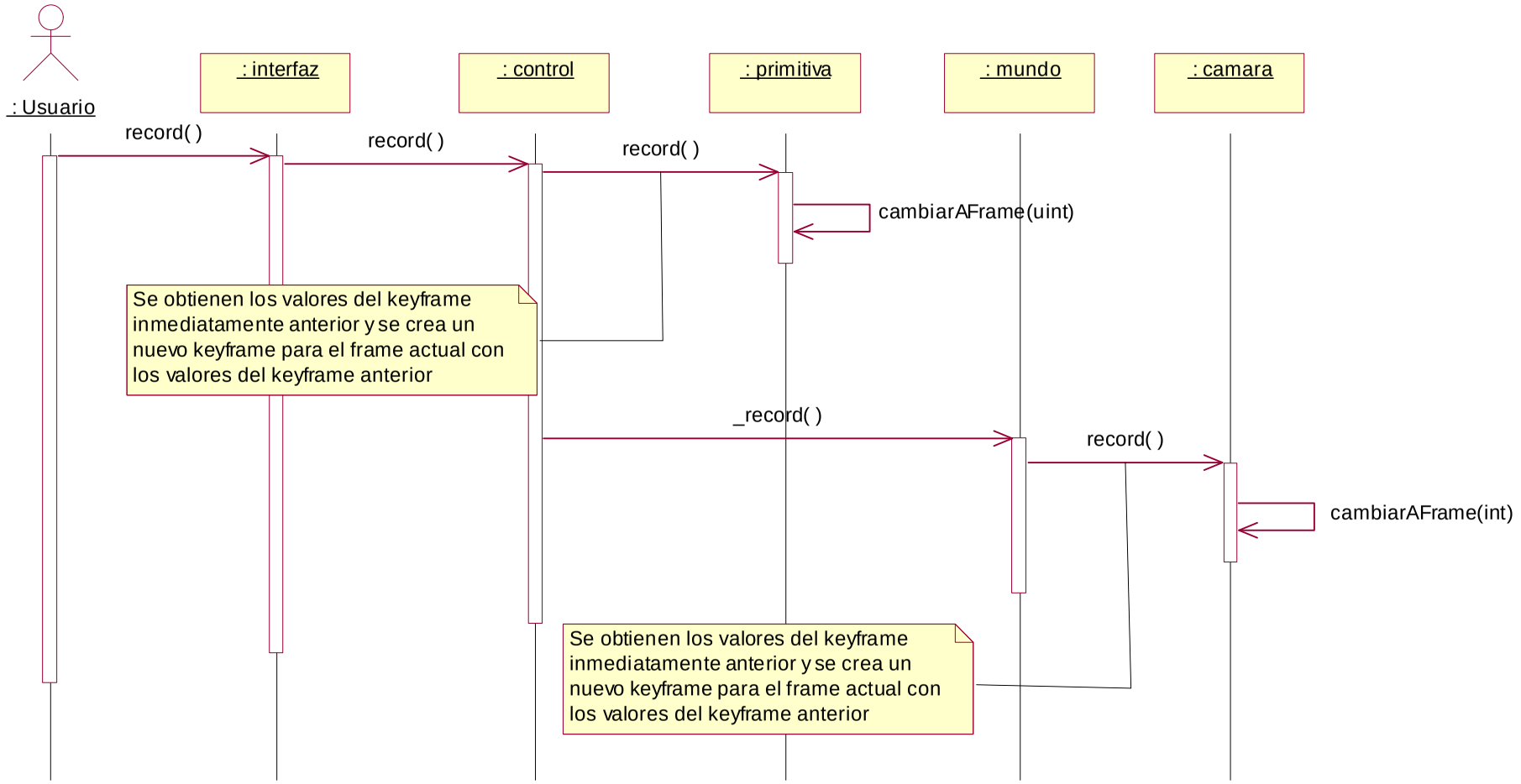
Caso de Uso: Desplazar a través de la animación



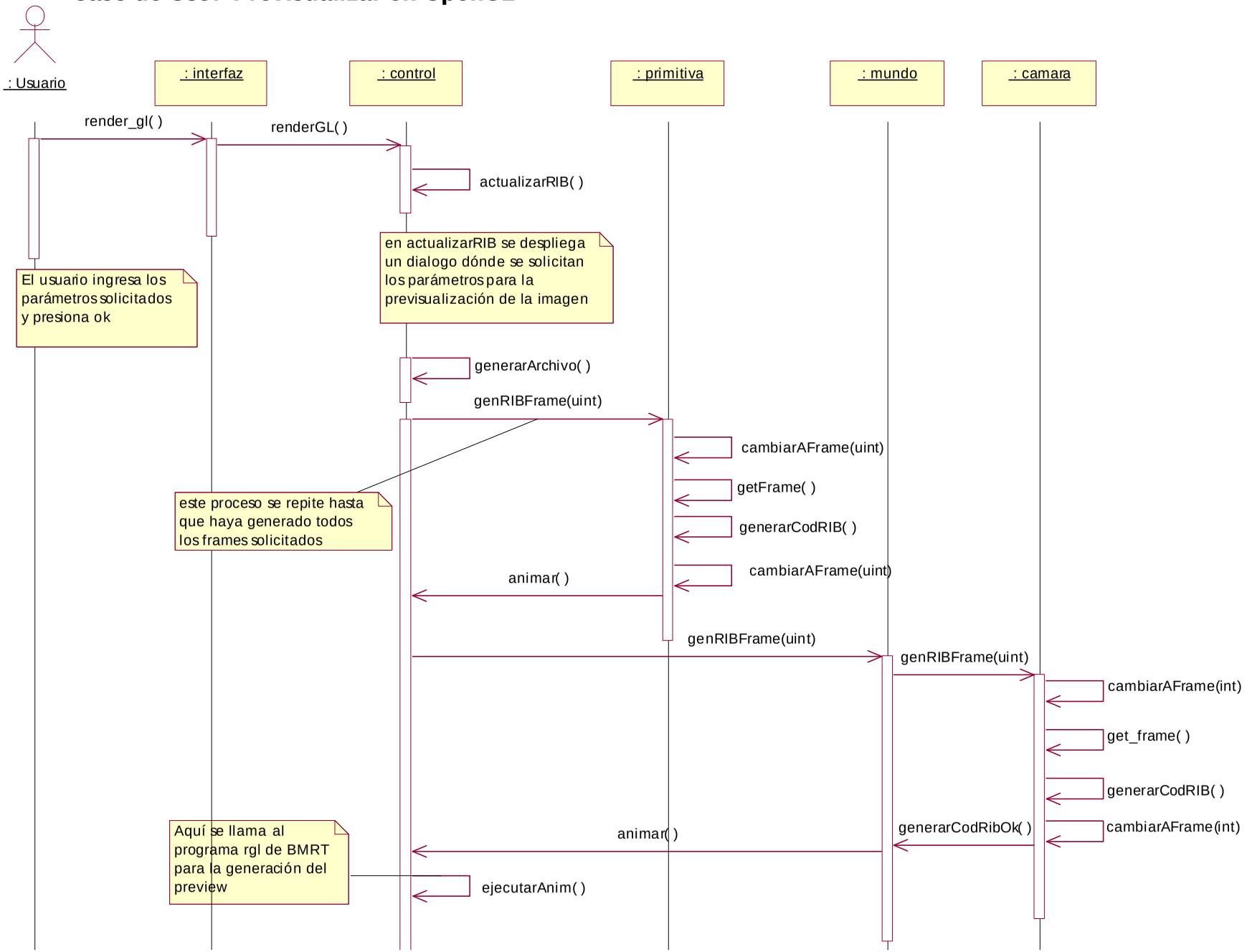
Caso de Uso: Definir tamaño y velocidad de animación



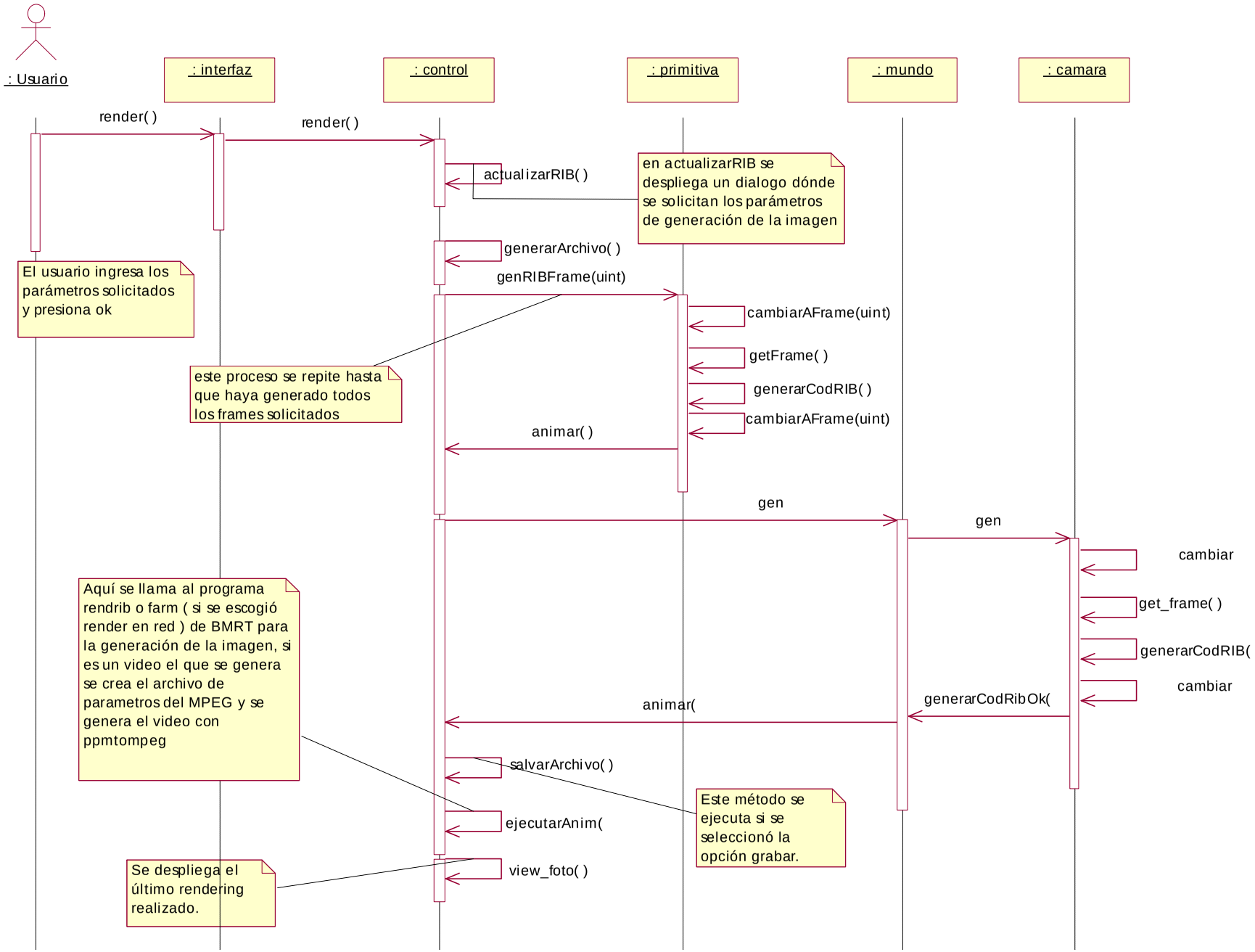
Caso de Uso: Crear Cuadro Clave (KeyFrame)



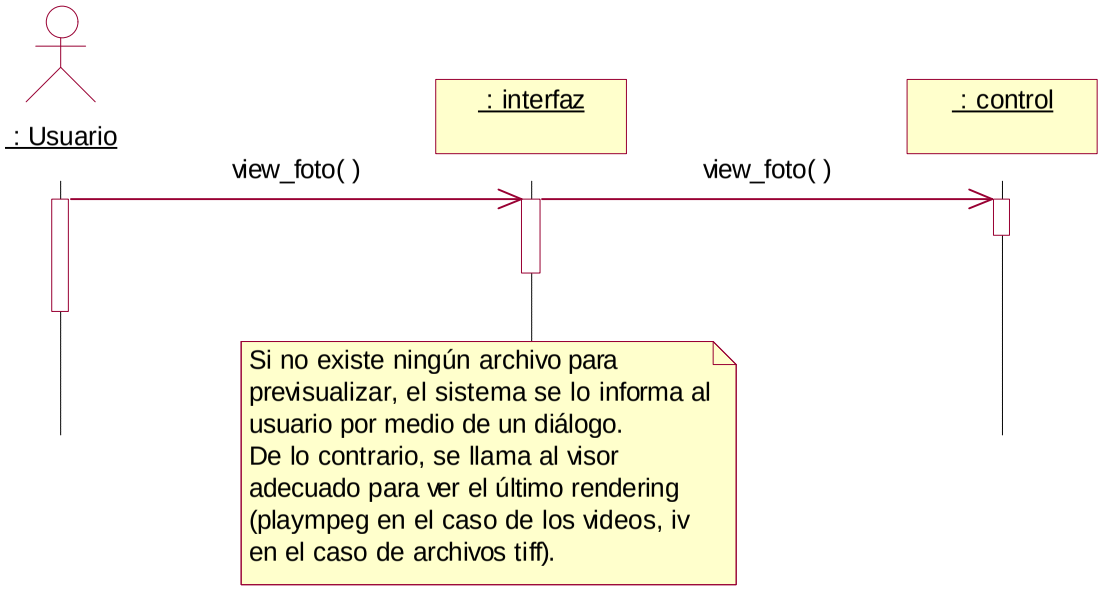
Caso de Uso: Previsualizar en OpenGL



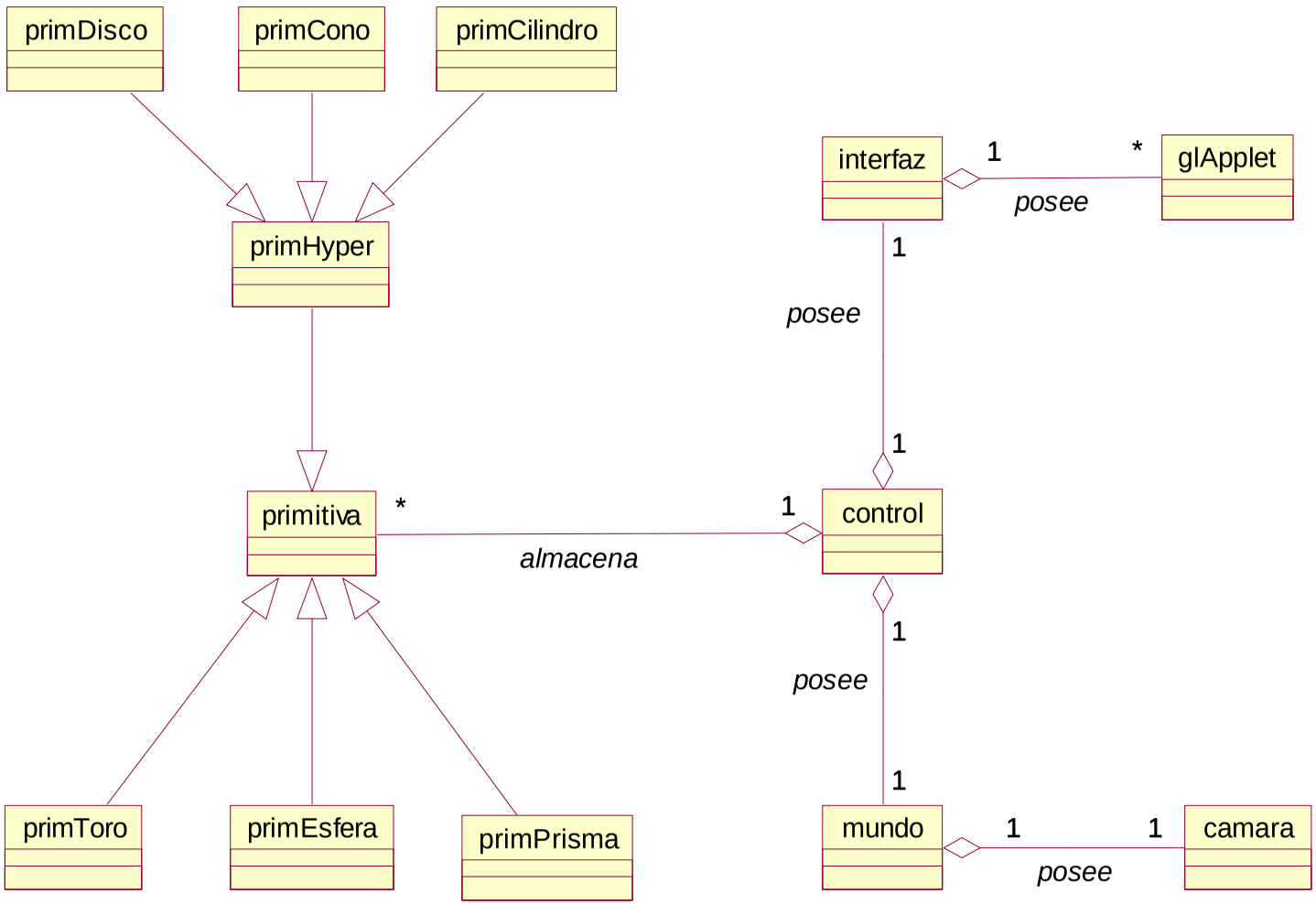
Caso de Uso: Generar Imagen o Vídeo Final



Caso de Uso: Desplegar Último Rendering



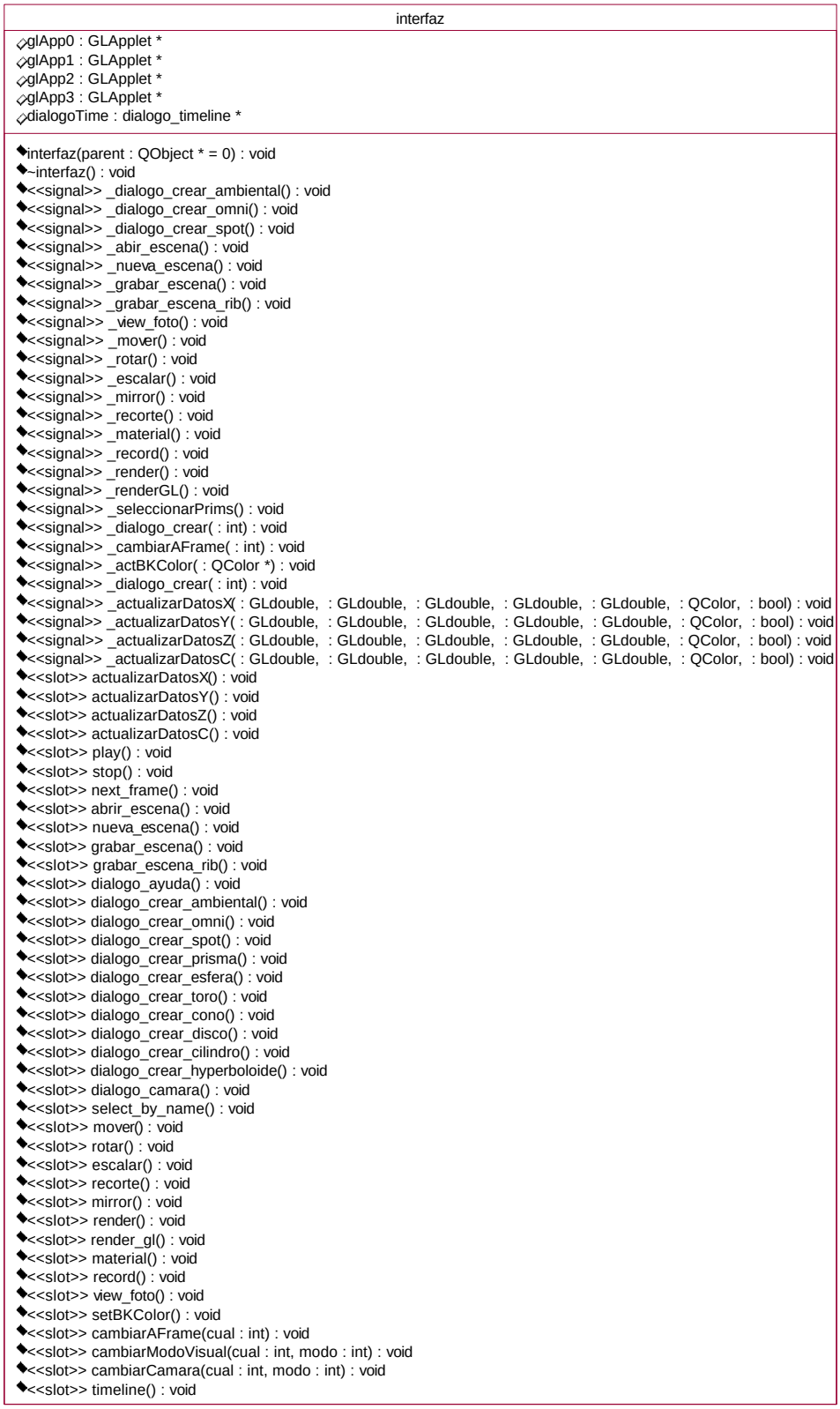
6.2. Diagrama de Estructura Estática

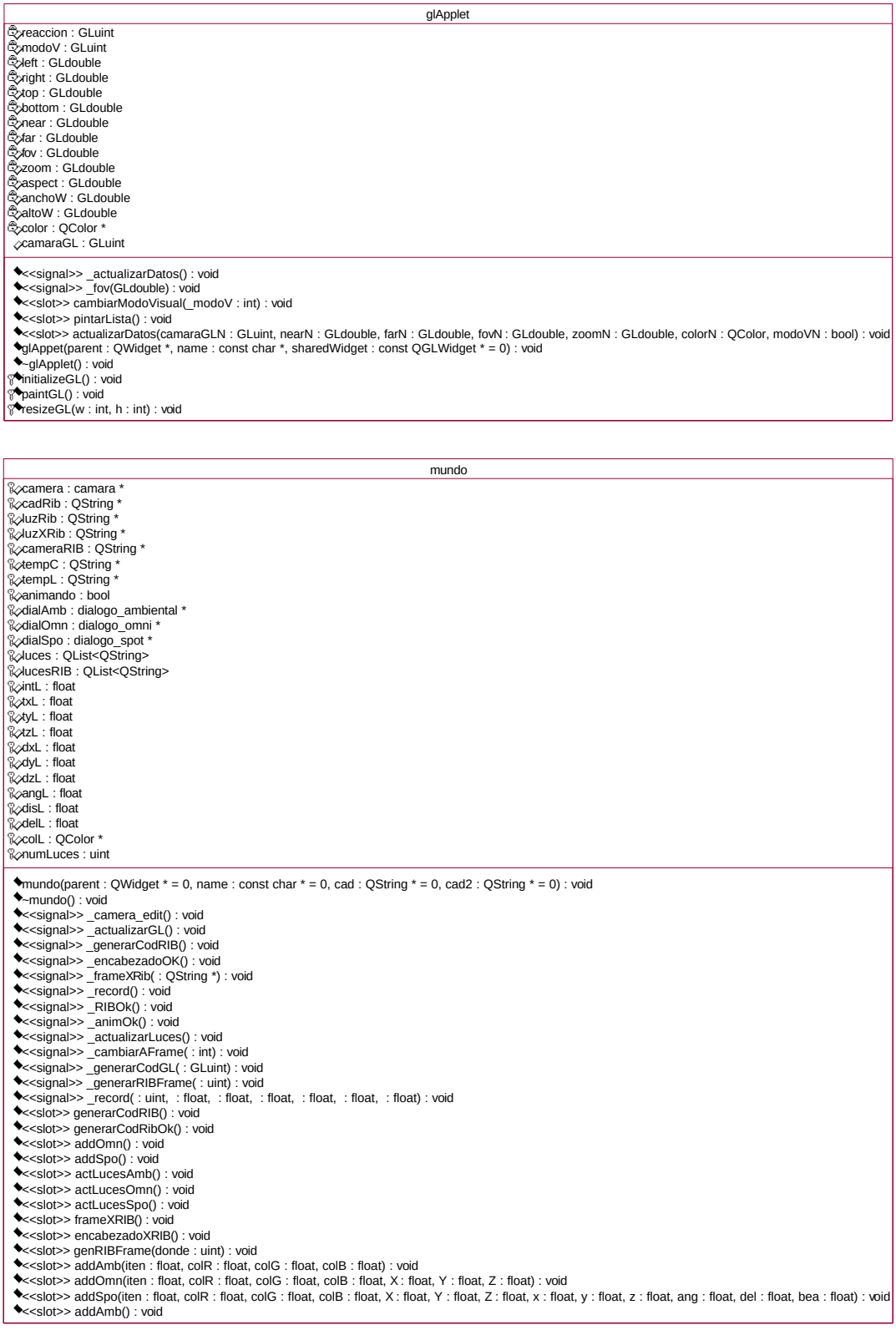


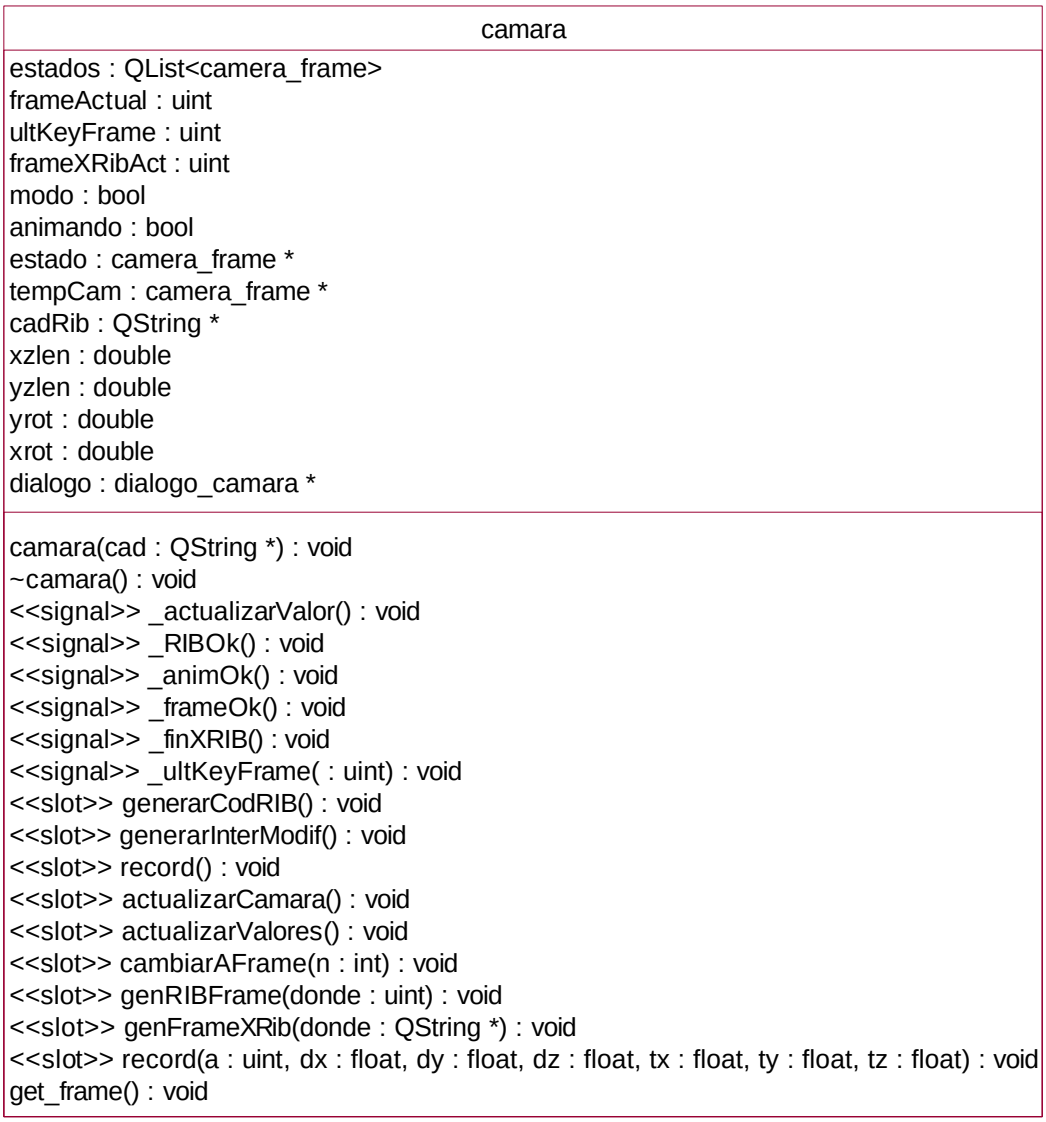
6.2.1. Diagrama de Estructura Estática (Detalle)

control

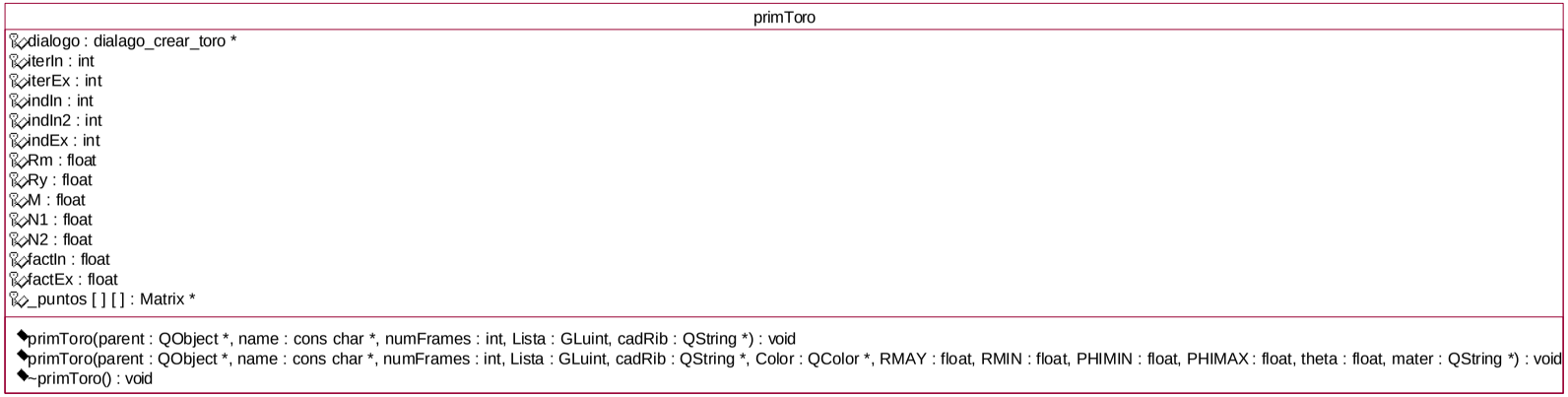
- control(parent : QObject *, name : const char *) : void
- ~control() : void
- <<signal>> _dialogo_crear_ambiental() : void
- <<signal>> _dialogo_crear_spot() : void
- <<signal>> _dialogo_crear_omni() : void
- <<signal>> _generarInterCreac() : void
- <<signal>> _rePaint() : void
- <<signal>> _actualizarGL() : void
- <<signal>> _actualizarRIB() : void
- <<signal>> _camera_edit() : void
- <<signal>> _finGenArch() : void
- <<signal>> _finGenArchSalvar() : void
- <<signal>> _obtEncabezadoXRib() : void
- <<signal>> _obtFramesXRib() : void
- <<signal>> _record() : void
- <<signal>> _inicializar() : void
- <<signal>> _resetPreview() : void
- <<signal>> _getNombres() : void
- <<signal>> _getNombresValidar() : void
- <<signal>> _nombresOK() : void
- <<signal>> _genRIBFrame(: uint) : void
- <<signal>> _baseGL(: GLuint) : void
- <<signal>> _cambiarAFrame(: int) : void
- <<signal>> _cambiarModoVisual(: int, : int) : void
- <<signal>> _cambiarCamara(: int, : int) : void
- <<signal>> _material(: QString *) : void
- <<signal>> _seleccionar(: QString, : bool) : void
- <<signal>> _dialogo_crear_ambiental(: float, : float, : float, : float) : void
- <<signal>> _dialogo_crear_omni(: float, : float, : float, : float, : float, : float, : float) : void
- <<signal>> _dialogo_crear_spot(: float, : float, : float, : float, : float, : float, : float, : float, : float, : float, : float, : float, : float) : void
- <<signal>> _record(: uint, : float, : float, : float, : float, : float, : float) : void
- <<signal>> _record(: Matrix, : uint, : QString *) : void
- <<signal>> _moverPreview(: Matrix) : void
- <<signal>> _moverTransf(: Matrix) : void
- <<signal>> _rotarPreview(: Matrix) : void
- <<signal>> _rotarTransf(: Matrix) : void
- <<signal>> _escalarPreview(: Matrix) : void
- <<signal>> _escalarTransf(: Matrix) : void
- <<signal>> _recortePreview(: Matrix) : void
- <<signal>> _recorteTransf(: Matrix) : void
- <<signal>> _mirrorPreview(: Matrix) : void
- <<signal>> _mirrorTransf(: Matrix) : void
- <<slot>> actualizarRIB() : void
- <<slot>> construirListaPrims() : void
- <<slot>> dialogoSeleccion() : void
- <<slot>> mostrarDialogoSeleccion() : void
- <<slot>> primCreada() : void
- <<slot>> primNoCreada() : void
- <<slot>> actualizarSeleccion() : void
- <<slot>> actualizarPrims() : void
- <<slot>> iniciar() : void
- <<slot>> record() : void
- <<slot>> mover() : void
- <<slot>> rotar() : void
- <<slot>> escalar() : void
- <<slot>> recorte() : void
- <<slot>> mirror() : void
- <<slot>> material() : void
- <<slot>> moverPreview() : void
- <<slot>> moverTransf() : void
- <<slot>> calcularMatrizMover() : void
- <<slot>> rotarPreview() : void
- <<slot>> rotarTransf() : void
- <<slot>> calcularMatrixRotar() : void
- <<slot>> escalarPreview() : void
- <<slot>> escalarTransf() : void
- <<slot>> calcularMatrixEscalar() : void
- <<slot>> recortePreview() : void
- <<slot>> recorteTransf() : void
- <<slot>> calcularMatrixRecorte() : void
- <<slot>> mirrorPreview() : void
- <<slot>> mirrorTransf() : void
- <<slot>> calcularMatrixMirror() : void
- <<slot>> render() : void
- <<slot>> renderGL() : void
- <<slot>> generarArchivo() : void
- <<slot>> ejecutarAnim() : void
- <<slot>> animar() : void
- <<slot>> view_foto() : void
- <<slot>> abrir_escena() : void
- <<slot>> nueva_escena() : void
- <<slot>> guardar_escena() : void
- <<slot>> salvarArchivo() : void
- <<slot>> generarXRIB() : void
- <<slot>> ensamblarXRIB() : void
- <<slot>> genEncabezadoRIB() : void
- <<slot>> guardar_escena_rib() : void
- <<slot>> dialogo_crear(cual : int) : void
- <<slot>> revisarNombre(nombre : const QString &) : void
- <<slot>> validarNombres(nombre : QString) : void
- <<slot>> addLista(nombre : QString) : void
- <<slot>> fov(dato : GLdouble) : void
- <<slot>> ultKeyFrame(valor : uint) : void
- <<slot>> actBKColor(Color : QColor *) : void


interfaz
◇glApp0 : GLApplet *
◇glApp1 : GLApplet *
◇glApp2 : GLApplet *
◇glApp3 : GLApplet *
◇dialogoTime : dialogo_timeline *
◆interfaz(parent : QObject * = 0) : void
◆~interfaz() : void
◆<<signal>> _dialogo_crear_ambiental() : void
◆<<signal>> _dialogo_crear_omni() : void
◆<<signal>> _dialogo_crear_spot() : void
◆<<signal>> _abir_escena() : void
◆<<signal>> _nueva_escena() : void
◆<<signal>> _grabar_escena() : void
◆<<signal>> _grabar_escena_rib() : void
◆<<signal>> _view_foto() : void
◆<<signal>> _mover() : void
◆<<signal>> _rotar() : void
◆<<signal>> _escalar() : void
◆<<signal>> _mirror() : void
◆<<signal>> _recorte() : void
◆<<signal>> _material() : void
◆<<signal>> _record() : void
◆<<signal>> _render() : void
◆<<signal>> _renderGL() : void
◆<<signal>> _seleccionarPrims() : void
◆<<signal>> _dialogo_crear(: int) : void
◆<<signal>> _cambiarAFrame(: int) : void
◆<<signal>> _actBKColor(: QColor *) : void
◆<<signal>> _dialogo_crear(: int) : void
◆<<signal>> _actualizarDatosX(: GLdouble, : GLdouble, : GLdouble, : GLdouble, : GLdouble, : QColor, : bool) : void
◆<<signal>> _actualizarDatosY(: GLdouble, : GLdouble, : GLdouble, : GLdouble, : GLdouble, : QColor, : bool) : void
◆<<signal>> _actualizarDatosZ(: GLdouble, : GLdouble, : GLdouble, : GLdouble, : GLdouble, : QColor, : bool) : void
◆<<signal>> _actualizarDatosC(: GLdouble, : GLdouble, : GLdouble, : GLdouble, : GLdouble, : QColor, : bool) : void
◆<<slot>> actualizarDatosX() : void
◆<<slot>> actualizarDatosY() : void
◆<<slot>> actualizarDatosZ() : void
◆<<slot>> actualizarDatosC() : void
◆<<slot>> play() : void
◆<<slot>> stop() : void
◆<<slot>> next_frame() : void
◆<<slot>> abrir_escena() : void
◆<<slot>> nueva_escena() : void
◆<<slot>> grabar_escena() : void
◆<<slot>> grabar_escena_rib() : void
◆<<slot>> dialogo_ayuda() : void
◆<<slot>> dialogo_crear_ambiental() : void
◆<<slot>> dialogo_crear_omni() : void
◆<<slot>> dialogo_crear_spot() : void
◆<<slot>> dialogo_crear_prisma() : void
◆<<slot>> dialogo_crear_esfera() : void
◆<<slot>> dialogo_crear_toro() : void
◆<<slot>> dialogo_crear_cono() : void
◆<<slot>> dialogo_crear_disco() : void
◆<<slot>> dialogo_crear_cilindro() : void
◆<<slot>> dialogo_crear_hyperboloide() : void
◆<<slot>> dialogo_camara() : void
◆<<slot>> select_by_name() : void
◆<<slot>> mover() : void
◆<<slot>> rotar() : void
◆<<slot>> escalar() : void
◆<<slot>> recorte() : void
◆<<slot>> mirror() : void
◆<<slot>> render() : void
◆<<slot>> render_gl() : void
◆<<slot>> material() : void
◆<<slot>> record() : void
◆<<slot>> view_foto() : void
◆<<slot>> setBKColor() : void
◆<<slot>> cambiarAFrame(cual : int) : void
◆<<slot>> cambiarModoVisual(cual : int, modo : int) : void
◆<<slot>> cambiarCamara(cual : int, modo : int) : void
◆<<slot>> timeline() : void

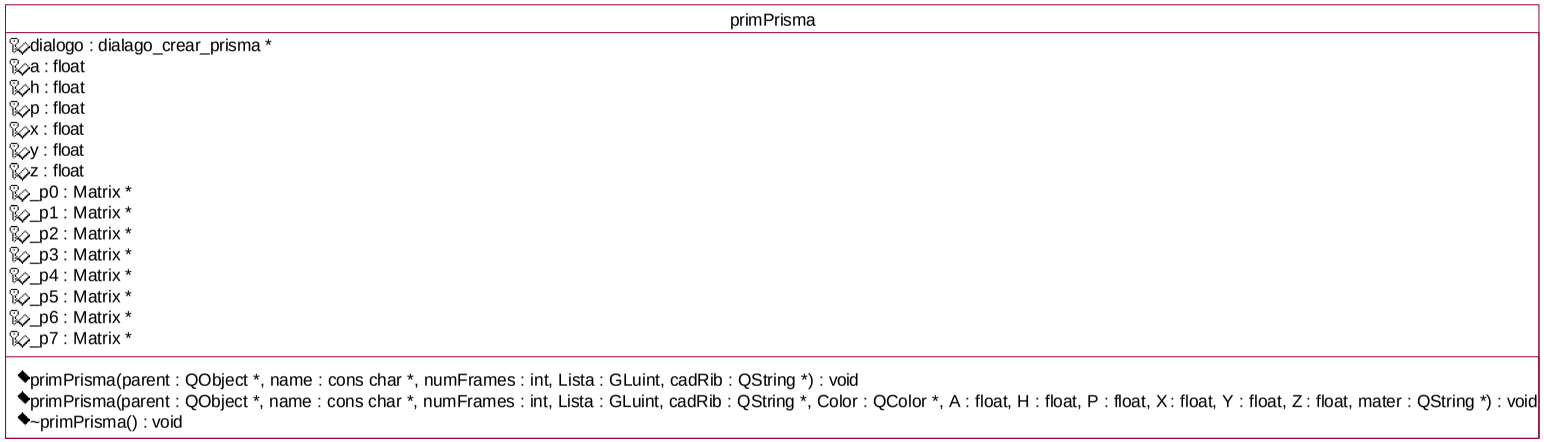

glApplet
reaccion : GLuint
modoV : GLuint
left : GLdouble
right : GLdouble
top : GLdouble
bottom : GLdouble
near : GLdouble
far : GLdouble
fov : GLdouble
zoom : GLdouble
aspect : GLdouble
anchoW : GLdouble
altoW : GLdouble
color : QColor *
camaraGL : GLuint
<<signal>> _actualizarDatos() : void
<<signal>> _fov(GLdouble) : void
<<slot>> cambiarModoVisual(_modoV : int) : void
<<slot>> pintarLista() : void
<<slot>> actualizarDatos(camaraGLN : GLuint, nearN : GLdouble, farN : GLdouble, fovN : GLdouble, zoomN : GLdouble, colorN : QColor, modoVN : bool) : void
glAppet(parent : QWidget *, name : const char *, sharedWidget : const QGLWidget * = 0) : void
~glApplet() : void
initializeGL() : void
paintGL() : void
resizeGL(w : int, h : int) : void
mundo
camera : camara *
cadRib : QString *
luzRib : QString *
luzXRib : QString *
cameraRIB : QString *
tempC : QString *
tempL : QString *
animando : bool
dialAmb : dialogo_ambiental *
dialOmn : dialogo_omni *
dialSpo : dialogo_spot *
luces : QList<QString>
lucesRIB : QList<QString>
intL : float
txL : float
tyL : float
tzL : float
dxL : float
dyL : float
dzL : float
angL : float
disL : float
delL : float
colL : QColor *
numLuces : uint
mundo(parent : QWidget * = 0, name : const char * = 0, cad : QString * = 0, cad2 : QString * = 0) : void
~mundo() : void
<<signal>> _camera_edit() : void
<<signal>> _actualizarGL() : void
<<signal>> _generarCodRIB() : void
<<signal>> _encabezadoOK() : void
<<signal>> _frameXRib(: QString *) : void
<<signal>> _record() : void
<<signal>> _RIBOk() : void
<<signal>> _animOk() : void
<<signal>> _actualizarLuces() : void
<<signal>> _cambiarAFrame(: int) : void
<<signal>> _generarCodGL(: GLuint) : void
<<signal>> _generarRIBFrame(: uint) : void
<<signal>> _record(: uint, : float, : float, : float, : float, : float, : float) : void
<<slot>> generarCodRIB() : void
<<slot>> generarCodRibOk() : void
<<slot>> addOmn() : void
<<slot>> addSpo() : void
<<slot>> actLucesAmb() : void
<<slot>> actLucesOmn() : void
<<slot>> actLucesSpo() : void
<<slot>> frameXRIB() : void
<<slot>> encabezadoXRIB() : void
<<slot>> genRIBFrame(donde : uint) : void
<<slot>> addAmb(iten : float, colR : float, colG : float, colB : float) : void
<<slot>> addOmn(iten : float, colR : float, colG : float, colB : float, X : float, Y : float, Z : float) : void
<<slot>> addSpo(iten : float, colR : float, colG : float, colB : float, X : float, Y : float, Z : float, x : float, y : float, z : float, ang : float, del : float, bea : float) : void
<<slot>> addAmb() : void

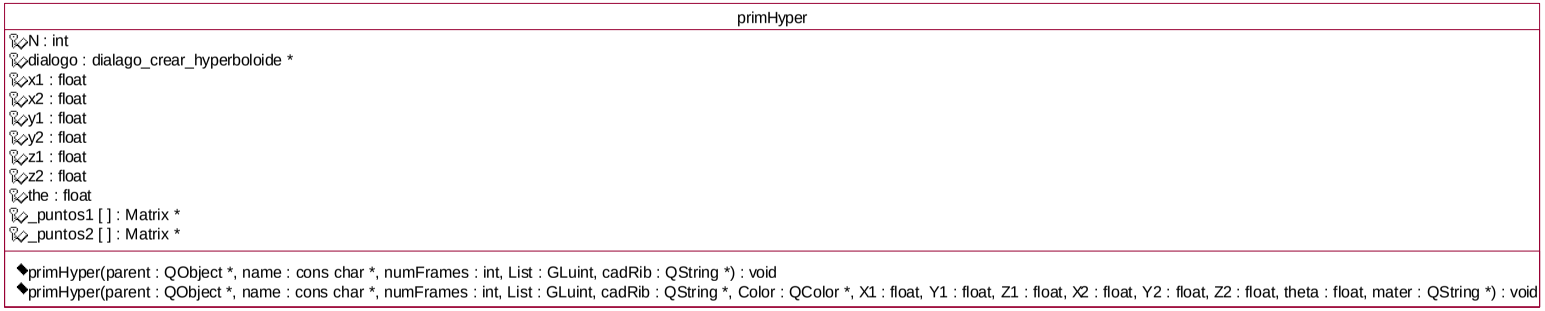

camara
estados : QList<camera_frame>
frameActual : uint
ultKeyFrame : uint
frameXRibAct : uint
modo : bool
animando : bool
estado : camera_frame *
tempCam : camera_frame *
cadRib : QString *
xzlen : double
yzlen : double
yrot : double
xrot : double
dialogo : dialogo_camara *
camara(cad : QString *) : void
~camara() : void
<<signal>> _actualizarValor() : void
<<signal>> _RIBOk() : void
<<signal>> _animOk() : void
<<signal>> _frameOk() : void
<<signal>> _finXRIB() : void
<<signal>> _ultKeyFrame(: uint) : void
<<slot>> generarCodRIB() : void
<<slot>> generarInterModif() : void
<<slot>> record() : void
<<slot>> actualizarCamara() : void
<<slot>> actualizarValores() : void
<<slot>> cambiarAFrame(n : int) : void
<<slot>> genRIBFrame(donde : uint) : void
<<slot>> genFrameXRib(donde : QString *) : void
<<slot>> record(a : uint, dx : float, dy : float, dz : float, tx : float, ty : float, tz : float) : void
get_frame() : void

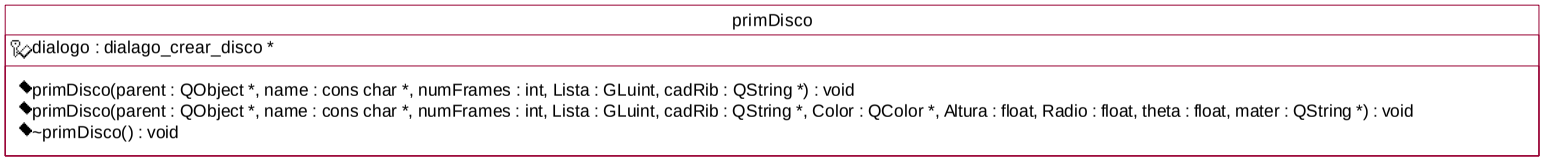

primitiva
desde : GLuint
hasta : GLuint
actual : GLuint
lista : GLuint
ultKeyFrame : uint
frameXRibAct : uint
keyF : bool
selecc : bool
noGen : bool
animando : bool
color : QColor *
cadRIB : QString *
nombre : QString *
matRib : QString *
preview : Matrix *
actualM : primFrame *
frames : QList<primFrame>
primitiva(parent : QObject *, name : cons char *, numFrames : int, cadRib : QString *) : void
~primitiva() : void
<<signal>> _objetoCreado() : void
<<signal>> _objetoNoCreado() : void
<<signal>> _actualizacionPrimOK() : void
<<signal>> _actualizarValor() : void
<<signal>> _RIBOk() : void
<<signal>> _animOk() : void
<<signal>> _encabezadoOK() : void
<<signal>> _frameOK() : void
<<signal>> _finXRIB() : void
<<signal>> _nombre(: QString) : void
<<signal>> _validarNombre(: QString) : void
<<signal>> _revisarNombre(: const QString &) : void
<<virtual slot>> generarCodGL() : void
<<virtual slot>> encabezadoXRIB() : void
<<virtual slot>> generarCodRIB() : void
<<virtual slot>> generarInterCreac() : void
<<virtual slot>> actualizarValores() : void
<<virtual slot>> recalcularValoresIniciales() : void
<<virtual slot>> validarNombre() : void
<<virtual slot>> genFrameXRib() : void
<<slot>> getNombre() : void
<<slot>> resetPreview() : void
<<slot>> record() : void
<<slot>> recalcularValores() : void
<<slot>> getFrame() : void
<<slot>> genRIBFrame(donde : uint) : void
<<slot>> cambiarAFrame(donde : uint) : void
<<slot>> material(cual : QString *) : void
<<slot>> addTransf(elem : Matrix) : void
<<slot>> addPreview(elem : Matrix) : void
<<slot>> addTransf(elem : Matrix, cualFrame : uint, queNombre : QString *) : void
<<slot>> seleccionar(nomb : QString, valor : bool) : void

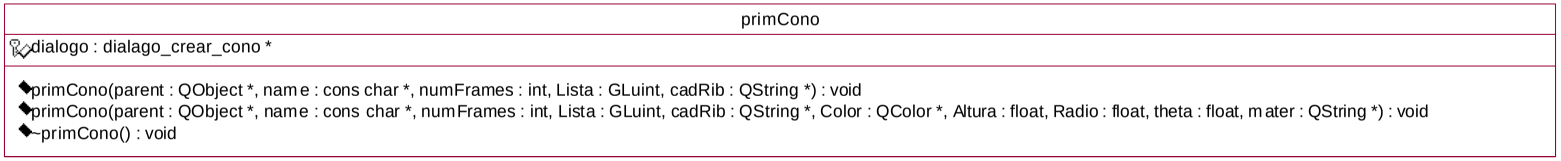

primToro
dialogo : dialago_crear_toro *
iterIn : int
iterEx : int
indIn : int
indIn2 : int
indEx : int
Rm : float
Ry : float
M : float
N1 : float
N2 : float
factIn : float
factEx : float
_puntos [ ] [ ] : Matrix *
primToro(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *) : void
primToro(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *, Color : QColor *, RMAY : float, RMIN : float, PHIMIN : float, PHIMAX : float, theta : float, mater : QString *) : void
~primToro() : void

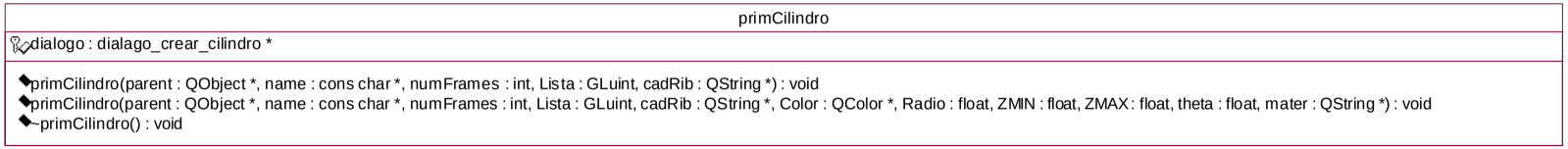

primEsfera
dialogo : dialago_crear_esfera *
iterIn : int
iterEx : int
indIn : int
indIn2 : int
indEx : int
R : float
Zmax : float
Zmin : float
the : float
factIn : float
factEx : float
_puntos1 [ ] [ ] : Matrix *
_puntos2 [ ] [ ] : Matrix *
_puntos3 [ ] [ ] : Matrix *
_puntos4 [ ] [ ] : Matrix *
_puntos5 [ ] [ ] : Matrix *
_puntos6 [ ] [ ] : Matrix *
_puntos7 [ ] [ ] : Matrix *
_puntos8 [ ] [ ] : Matrix *
primEsfera(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *) : void
primEsfera(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *, Color : QColor *, Radio : float, ZMIN : float, ZMAX : float, theta : float, mater : QString *) : void
~primToro() : void


primPrisma
dialogo : dialago_crear_prisma *
a : float
h : float
p : float
x : float
y : float
z : float
_p0 : Matrix *
_p1 : Matrix *
_p2 : Matrix *
_p3 : Matrix *
_p4 : Matrix *
_p5 : Matrix *
_p6 : Matrix *
_p7 : Matrix *
primPrisma(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *) : void
primPrisma(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *, Color : QColor *, A : float, H : float, P : float, X : float, Y : float, Z : float, mater : QString *) : void
~primPrisma() : void


primHyper
N : int
dialogo : dialago_crear_hyperboloide *
x1 : float
x2 : float
y1 : float
y2 : float
z1 : float
z2 : float
the : float
_puntos1 [ ] : Matrix *
_puntos2 [ ] : Matrix *
primHyper(parent : QObject *, name : cons char *, numFrames : int, List : GLuint, cadRib : QString *) : void
primHyper(parent : QObject *, name : cons char *, numFrames : int, List : GLuint, cadRib : QString *, Color : QColor *, X1 : float, Y1 : float, Z1 : float, X2 : float, Y2 : float, Z2 : float, theta : float, mater : QString *) : void


primDisco
dialogo : dialago_crear_disco *
primDisco(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *) : void
primDisco(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *, Color : QColor *, Altura : float, Radio : float, theta : float, mater : QString *) : void
~primDisco() : void


primCono
dialogo : dialago_crear_cono *
primCono(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *) : void
primCono(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *, Color : QColor *, Altura : float, Radio : float, theta : float, mater : QString *) : void
~primCono() : void


primCilindro
dialogo : dialago_crear_cilindro *
primCilindro(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *) : void
primCilindro(parent : QObject *, name : cons char *, numFrames : int, Lista : GLuint, cadRib : QString *, Color : QColor *, Radio : float, ZMIN : float, ZMAX : float, theta : float, mater : QString *) : void
~primCilindro() : void

6.3 Diagramas de Colaboración

Diagrama de colaboración: Grabar Escena

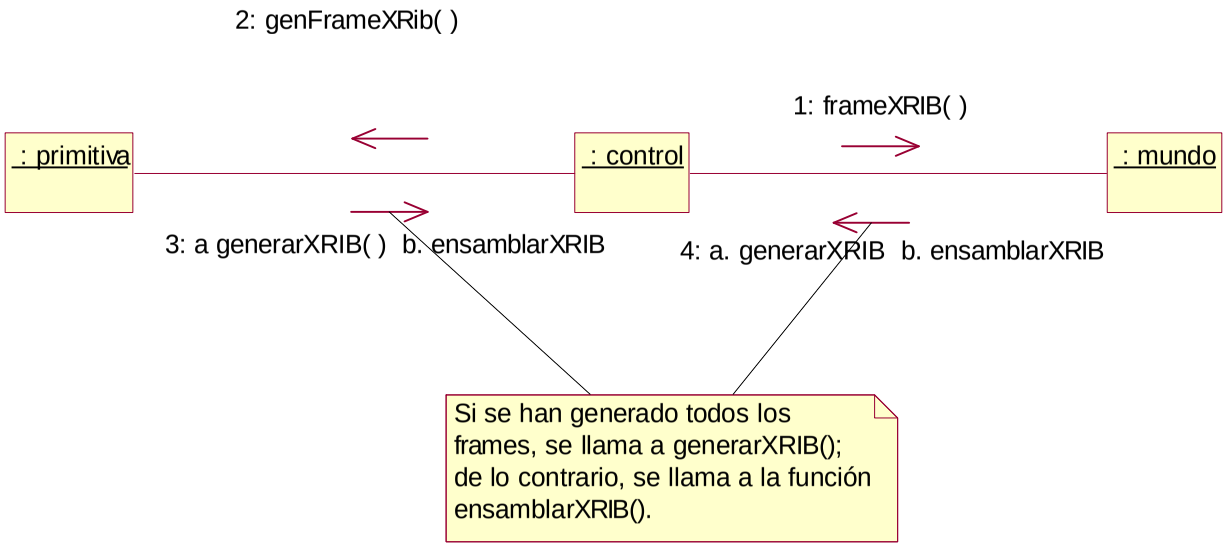


Diagrama de colaboración: Guardar Como

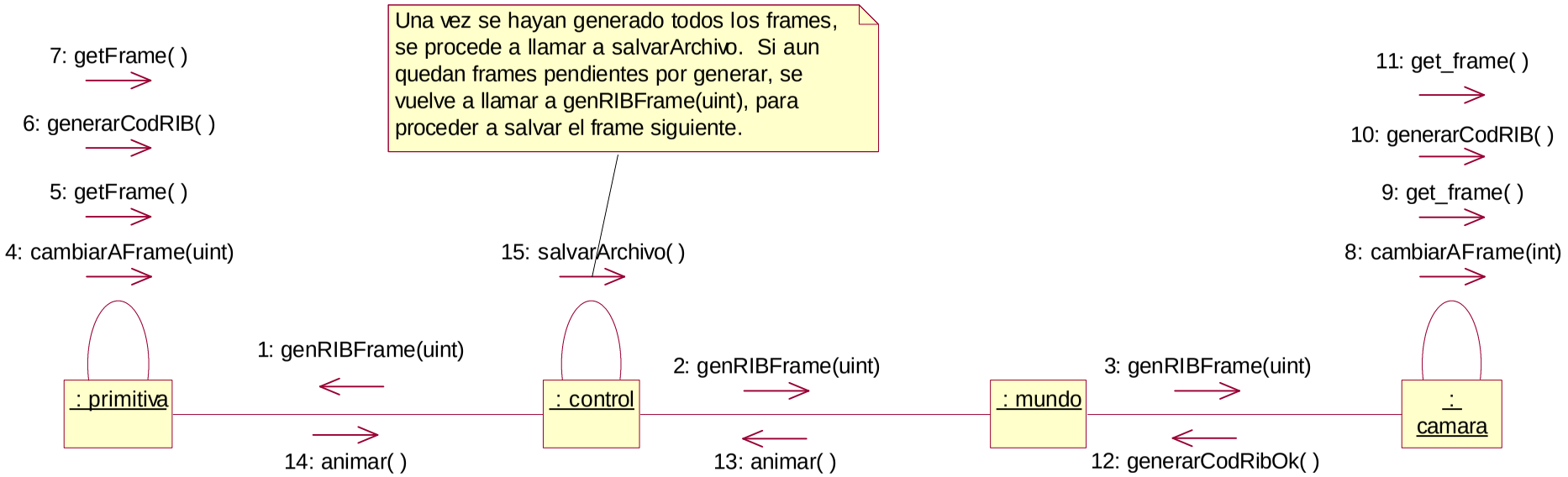


Diagrama de colaboración: Seleccionar Objetos

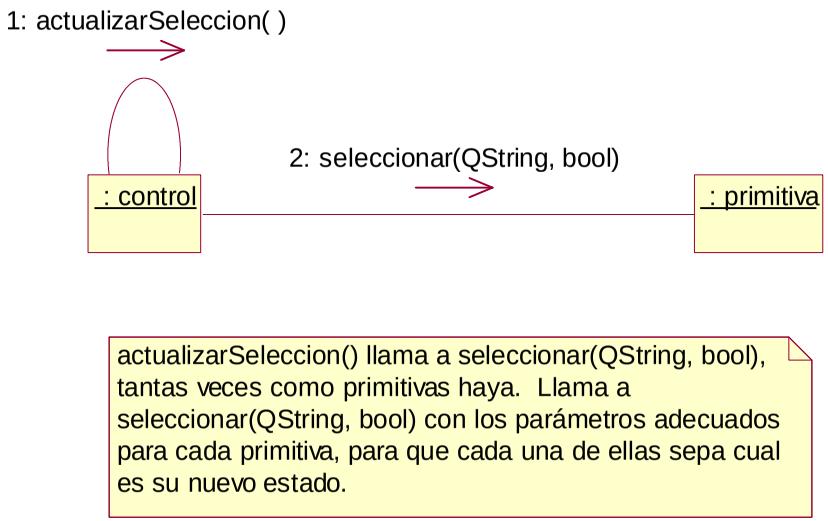


Diagrama de colaboración: Generar a Imagen o Vídeo final , método Cambiar a Frame

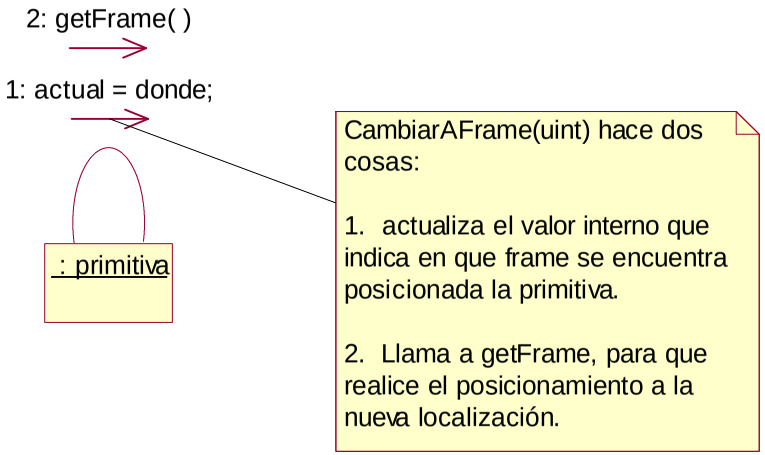
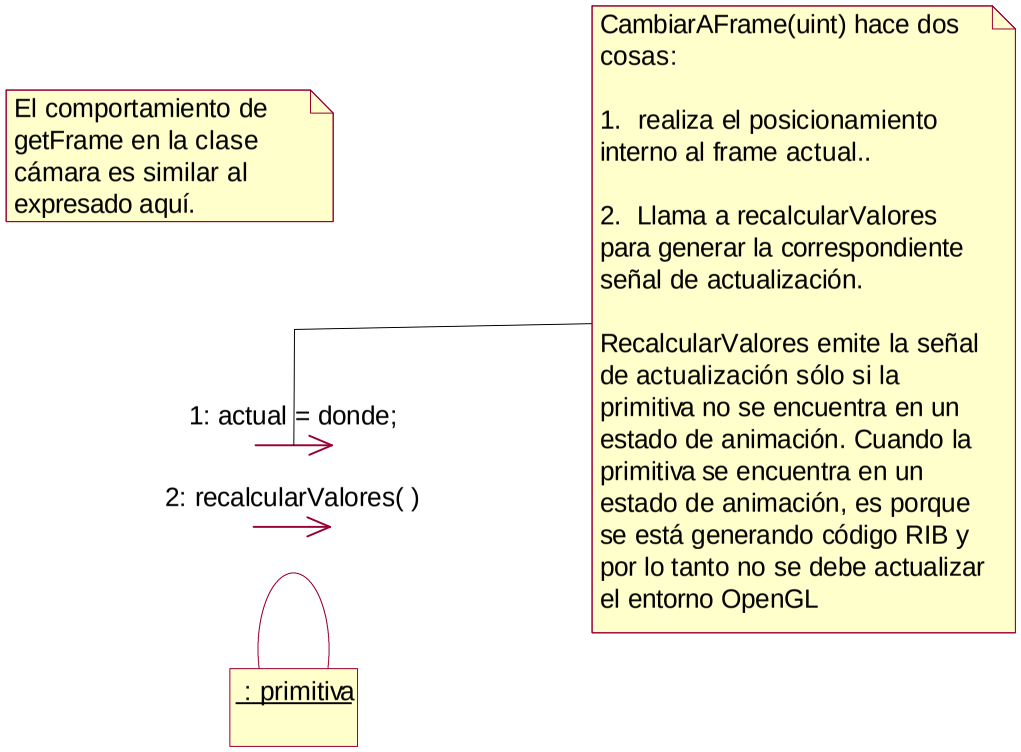


Diagrama de colaboración: Generar imagen o video final, método getFrame



6.4 Contratos

Nombre	**guardar_escena**
Responsabilidades	Crear un stream temporal de salida y emitir la señal _obtEncabezadoXRib que está conectada con encabezadoXRIB de primitiva y con encabezadoXRIB de mundo
Clase	control
Referencias Cruzadas	Requerimientos funcionales: 31,32,33,34 Casos de uso: Grabar escena, Cerrar escena, Abrir escena, Crear escena
Tipo	slot
Notas	Este slot se activa cuando recibe la señal _grabar_escena de interfaz
Salida	
Precondiciones	
Poscondiciones	Control ha emitido la señal _obtEncabezadoXRib y se ha creado el stream de salida.

Nombre	**encabezadoXRIB**
Responsabilidades	Ingresa al stream de salida los datos correspondientes a las luces y emite la señal _encabezadoOK
Clase	mundo
Referencias Cruzadas	Requerimientos funcionales: 32 Casos de uso: Grabar escena

Tipo	slot
Notas	Este slot se activa cuando recibe la señal _obtEncabezadoXRib de control
Salida	
Precondiciones	El stream de salida se encuentra creado
Poscondiciones	mundo ha emitido la señal _encabezadoOK y se ha modificado el stream de salida.

Nombre	**encabezadoXRIB**
Responsabilidades	Ingresa al stream de salida los datos correspondientes a la primitiva y emite la señal _encabezadoOK
Clase	primitiva
Referencias Cruzadas	Requerimientos funcionales: 32 Casos de uso: Grabar escena
Tipo	slot
Notas	Este slot se activa cuando recibe la señal _obtEncabezadoXRib de control
Salida	
Precondiciones	El stream de salida se encuentra creado
Poscondiciones	primitiva ha emitido la señal _encabezadoOK y se ha modificado el stream de salida.

Nombre	**ensamblarXRIB**
Responsabilidades	Verificar si ya los elementos de la escena mundo y primitiva en el frame actual han escrito en el stream de salida su información y

	de ser así emitir _obtFramesXRib.
Clase	control
Referencias Cruzadas	Requerimientos funcionales: 32 Casos de uso: Grabar escena
Tipo	slot
Notas	Este slot se activa cuando recibe la señal _frameOk de mundo y primitiva.
Salida	
Precondiciones	El frame procesado actualmente no sea el último.
Poscondiciones	Control ha emitido la señal _obtFramesXRib y se ha actualizado el stream de salida.

Nombre	**generarXRIB**
Responsabilidades	Verificar si ya los elementos de la escena mundo y primitiva en el último frame han escrito en el stream de salida su información y de ser así despliega un dialogo preguntando el nombre del archivo al usuario y copia el stream al archivo especificado.
Clase	control
Referencias Cruzadas	Requerimientos funcionales: 32 Casos de uso: Grabar escena
Tipo	slot
Notas	Este slot se activa cuando recibe la señal _finXRIB de mundo y primitiva.
Salida	
Precondiciones	El frame procesado actualmente sea el último.

Poscondiciones	Se ha almacenado el stream al archivo especificado por el usuario.

Nombre	**nueva_escena**
Responsabilidades	Liberar la memoria de la escena actual consultando antes si se desea guardar la misma y construir una nueva escena en blanco.
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 34, 32 Casos de uso: Crear escena, Cerrar escena
Tipo	Slot
Notas	Este slot se activa cuando recibe la señal _nueva_escena de interfaz
Salida	
Precondiciones	
Poscondiciones	Se ha creado una nueva escena.

Nombre	**Iniciar**
Responsabilidades	Reinicializa las listas de OpenGL y emite las señales de reinicialización de los paneles de OpenGl.
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 34, 32 Casos de uso: Crear escena, Cerrar escena
Tipo	slot
Notas	

Salida	
Precondiciones	
Poscondiciones	La información de OpenGL ha sido reinicializada

Nombre	**abrir_escena**
Responsabilidades	Consultar si se desea guardar la escena actual, desplegar un dialogo para seleccionar el archivo a cargar y posteriormente carga el archivo tipo Xrib
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 31, 32 Casos de uso: Abrir escena
Tipo	slot
Notas	Este slot es activado cuando se emite la señal _abrir_escena en interfaz.
Salida	
Precondiciones	
Poscondiciones	Se ha cargado una escena

Nombre	**grabar_escena_rib**
Responsabilidades	Desplegar un dialogo para seleccionar el archivo a grabar llama actualizarRIB y generarArchivo para construir el archivo y guardarlo respectivamente.
Clase	control
Referencias Cruzadas	Requerimientos funcionales: 35

	Casos de uso: Guardar como.
Tipo	slot
Notas	Este slot es activado cuando se emite la señal _grabar_escena_rib en interfaz.
Salida	
Precondiciones	
Poscondiciones	Se ha salvado la escena en un archivo formato rib.

Nombre	**actualizarRib**
Responsabilidades	Crear el stream temporal de salida, posteriormente configurar los parámetros para ejecutar alguna de las siguientes acciones: salvar el archivo, generar preview OpenGL o renderizar la escena. Además llamar generarArchivo
Clase	control
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final.
Tipo	Slot
Notas	Este slot es activado cuando se emite la señal _actualir en interfaz.
Salida	
Precondiciones	
Poscondiciones	Se ha generado el stream temporal y configurado los parámetros para salvar o

	previsualizar o generar la escena. Se ha llamado a generarArchivo.

Nombre	**generarArchivo**
Responsabilidades	Construir el encabezado del archivo Rib y emitir _genRIBFrame(uint)
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha generado el encabezado del archivo Rib y se ha iniciado el proceso de generación y almacenamiento del mismo.

Nombre	**GenRIBFrame**
Responsabilidades	emitir _genRIBFrame(uint)
Clase	Mundo
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final.
Tipo	Slot
Notas	Este slot es activado cuando se emite la señal _genRIBFrame en control.

Salida	
Precondiciones	
Poscondiciones	Se ha emitido la señal _genRIBFrame(uint)

Nombre	**GenRIBFrame**
Responsabilidades	Generar el código Rib correspondiente al frame seleccionado. Emitir _animOK()
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final.
Tipo	Slot
Notas	Este slot es activado cuando se emite la señal _genRIBFrame en control.
Salida	
Precondiciones	
Poscondiciones	Se ha generado el código Rib correspondiente al frame indicado y se ha emitido la señal _animOK()

Nombre	**GenRIBFrame**
Responsabilidades	Generar el código Rib correspondiente al frame seleccionado.
Clase	Cámara
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final.

Tipo	Slot
Notas	Este slot es activado cuando se emite la señal _genRIBFrame en mundo.
Salida	
Precondiciones	
Poscondiciones	Se ha generado el código Rib correspondiente al frame indicado.

Nombre	**CambiarAFrame**
Responsabilidades	Cambiar el indicador del frame actual de la primitiva al especificado y llamar a getFrame.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29, 25, 27, 24 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final, Desplazar a Través de la Animación, Crear Cuadro Clave, Visualizar Cuadro de la Escena.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha cambiado el indicador interno del frame actual.

Nombre	**CambiarAFrame**
Responsabilidades	Cambiar el indicador del frame actual de la

	camara al especificado y llamar a getFrame.
Clase	Camara
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29, 25, 27, 24 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final, Desplazar a Través de la Animación, Crear Cuadro Clave, Visualizar Cuadro de la Escena.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha cambiado el indicador interno del frame actual.

Nombre	**getFrame**
Responsabilidades	Cambia el frame actual al especificado por el indicador interno.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29, 24 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final, Visualizar Cuadro de la Escena.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha cambiado el frame actual.

Nombre	**GetFrame**
Responsabilidades	Cambia el frame actual al especificado por el indicador interno.
Clase	Camara
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29, 24 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final, Visualizar Cuadro de la Escena.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha cambiado el frame actual.

Nombre	**generarCodRIB**
Responsabilidades	Genera el código Rib correspondiente al frame actual y lo almacena en su cadena interna de código Rib.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final
Tipo	Slot
Notas	
Salida	
Precondiciones	

Poscondiciones	Se ha generado el código Rib correspondiente al frame actual.

Nombre	**generarCodRIB**
Responsabilidades	Genera el código Rib correspondiente al frame actual y lo almacena en su cadena interna de código Rib.
Clase	Camara
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha generado el código Rib correspondiente al frame actual.

Nombre	**GenerarCodRIBOk**
Responsabilidades	Combina el código Rib correspondiente al frame actual de la cámara y al estado del mundo. Emitir _animOK o _RIBOk según sea si se encuentra animando o no.
Clase	Mundo
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final

Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha generado el código Rib correspondiente al entorno actual. Se ha emitido la señal adecuada _animOk o _RIBOk.

Nombre	**animar**
Responsabilidades	Construye el archivo Rib de acuerdo a los parámetros del rango de frames que debe incluir el mismo. Emitir _finGenArchSalvar si se está guardando la escena, _finGenArch si se está renderizando la escena o _genRIBFrame si aun no se ha terminado de generar todo el archivo.
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 35, 28, 29 Casos de uso: Guardar como, Previsualizar en OpenGL, Generar Imagen o Video Final
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha construido el archivo Rib indicado y se ha emitido la señal adecuada al proceso actual.

Nombre	**generarCodGL**

Responsabilidades	Genera la lista OpenGL correspondiente al estado actual de la primitiva.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 2, 3, 4, 5, 6, 7, 8, 12, 13, 24 Casos de uso: Crear Caja, Crear Esfera, Crear Cono, Crear Cilindro, Crear Disco, Crear Toroide, Crear Hyperboloide, Posicionar Cámara, Seleccionar Objetos, Visualizar Cuadro de la Escena
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha generado la lista OpenGL correspondiente al estado actual.

Nombre	**ActualizarPrims**
Responsabilidades	Actualiza la lista OpenGL de las primitivas y emite _RePaint.
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 2, 3, 4, 5, 6, 7, 8, 12, 13, 24 Casos de uso: Crear Caja, Crear Esfera, Crear Cono, Crear Cilindro, Crear Disco, Crear Toroide, Crear Hyperboloide, Posicionar Cámara, Seleccionar Objetos, Visualizar Cuadro de la Escena

Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha generado la lista OpenGL correspondiente a las primitivas.

Nombre	**GenerarInterCreac**
Responsabilidades	Genera la interfaz de creación apropiada a la primitiva, actualiza los valores de acuerdo a la misma y almacena el resultado de crear la primitiva. Emite _objetoCreado u _objetoNoCreado de acuerdo a la entrada por parte del usuario.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 2, 3, 4, 5, 6, 7, 8 Casos de uso: Crear Caja, Crear Esfera, Crear Cono, Crear Cilindro, Crear Disco, Crear Toroide, Crear Hyperboloide.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha modificado la primitiva de acuerdo a los parámetros del diálogo de creación.

Nombre	**primCreada**

Responsabilidades	Almacena la primitiva que se encuentra en proceso de creación, emitiendo _actualizarGL y _rePaint.
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 2, 3, 4, 5, 6, 7, 8 Casos de uso: Crear Caja, Crear Esfera, Crear Cono, Crear Cilindro, Crear Disco, Crear Toroide, Crear Hyperboloide.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha creado completamente una nueva primitiva.

Nombre	**generarInterModif**
Responsabilidades	Despliega un diálogo para actualizar los parámetros de la cámara, actualizando cuando se es necesario.
Clase	Cámara
Referencias Cruzadas	Requerimientos funcionales: 12 Casos de uso: Posicionar Cámara.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha almacenado la nueva posición de la cámara en el frame actual.

Nombre	**actualizarValores**
Responsabilidades	Actualiza los valores de la cámara y emite _actualizarValor.
Clase	Cámara
Referencias Cruzadas	Requerimientos funcionales: 12 Casos de uso: Posicionar Cámara.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha actualizado la nueva posición de la cámara en el frame actual.

Nombre	**AddAmb**
Responsabilidades	Despliega un diálogo para modificar los parámetros de creación de la luz ambiental.
Clase	Mundo
Referencias Cruzadas	Requerimientos funcionales: 9, 31 Casos de uso: Crear Luz Ambiental, Abrir Escena.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha actualizado los parámetros de la luz

	ambiental.
Nombre	**AddOmn**
Responsabilidades	Despliega un diálogo para agregar una nueva luz omnidireccional, mostrando los parámetros de creación de la misma.
Clase	Mundo
Referencias Cruzadas	Requerimientos funcionales: 10, 31 Casos de uso: Crear Luz Omnidireccional, Abrir Escena.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha creado una nueva luz omnidireccional, basado en los parámetros de la misma.

Nombre	**AddSpo**
Responsabilidades	Despliega un diálogo para agregar una nueva luz direccional, mostrando los parámetros de creación de la misma.
Clase	Mundo
Referencias Cruzadas	Requerimientos funcionales: 11, 31 Casos de uso: Crear Luz Direccional, Abrir Escena.
Tipo	Slot
Notas	
Salida	

Precondiciones	
Poscondiciones	Se ha creado una nueva luz Direccional, basado en los parámetros de la misma.

Nombre	**AddTransf**
Responsabilidades	Multiplica una matriz de transformación por la matriz actual.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 14, 15, 16, 17, 18, 31 Casos de uso: Trasladar, Rotar, Escalar, Cortar, Reflejar, abrirEscena
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha creado añadido una transformación al frame actual.

Nombre	**actualizarSeleccion**
Responsabilidades	Emite _seleccionar y _actualizarGL, para actualizar los datos de las primitivas que están siendo seleccionadas.
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 13 Casos de uso: Seleccionar Objetos.
Tipo	Slot

Notas	
Salida	
Precondiciones	
Poscondiciones	Se han emitido las señales apropiadas de selección.

Nombre	**seleccionar**
Responsabilidades	Actualiza el estado de selección de la primitiva.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 13 Casos de uso: Seleccionar Objetos.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha actualizado el estado de selección de la primitiva.

Nombre	**ActualizarDatos**
Responsabilidades	Actualiza los datos básicos del glApplet.
Clase	GlApplet
Referencias Cruzadas	Requerimientos funcionales: 21, 23 Casos de uso: Definir las Propiedades del Panel de Visualización, Realizar Acercamientos.
Tipo	Slot
Notas	
Salida	

Precondiciones	
Poscondiciones	Se ha actualizado los datos del objeto.

Nombre	**actualizarDatos**
Responsabilidades	Actualiza los datos básicos del glApplet.
Clase	glApplet
Referencias Cruzadas	Requerimientos funcionales: 21, 23 Casos de uso: Definir las Propiedades del Panel de Visualización, Realizar Acercamientos.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha actualizado los datos del objeto.

Nombre	**RecalcularValores**
Responsabilidades	Emite _actualizarValor, si no se encuentra animando.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 24 Casos de uso: Visualizar Cuadro de la Escena.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha actualizado los datos del objeto.

Nombre	**next_frame**
Responsabilidades	Llama a intervalos fijos de tiempo, a stepUp y a next_frame.
Clase	Interfaz
Referencias Cruzadas	Requerimientos funcionales: 25 Casos de uso: Desplazar a Través de la Animación.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha avanzado un frame de la escena.

Nombre	**Record**
Responsabilidades	Se crea un keyFrame para el frame actual, copiando sus valores del keyFrame inmediatamente anterior.
Clase	Primitiva
Referencias Cruzadas	Requerimientos funcionales: 27 Casos de uso: Crear Cuadro Clave.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha añadido un cuadro clave a la primitiva.

Nombre	**Record**

Responsabilidades	Se crea un keyFrame para el frame actual, copiando sus valores del keyFrame inmediatamente anterior.
Clase	Camara
Referencias Cruzadas	Requerimientos funcionales: 27 Casos de uso: Crear Cuadro Clave.
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha añadido un cuadro clave a la camara.

Nombre	**ejecutarAnim**
Responsabilidades	Renderiza un frame o crea una animación, bien sea grabando los resultados (tiff, MPEG) o solamente visualizando los resultados (preview con rgl).
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 28, 29 Casos de uso: Previsualizar en OpenGL, Generar Imagen o Video Final
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha añadido un cuadro clave a la camara.

Nombre	**view_foto**
Responsabilidades	Muestra los resultados de la última acción en la que se generaron imágenes tiff o videos mpeg.
Clase	Control
Referencias Cruzadas	Requerimientos funcionales: 29, 30 Casos de uso: Generar Imagen o Video Final, Desplegar Último Render
Tipo	Slot
Notas	
Salida	
Precondiciones	
Poscondiciones	Se ha mostrado el último render persistente, realizado.

7. CONSTRUCCIÓN DE LA HERRAMIENTA DE SOFTWARE

A continuación se presentan las diferentes ventanas desarrolladas para la aplicación cuya utilización está detallada en el manual de usuario.

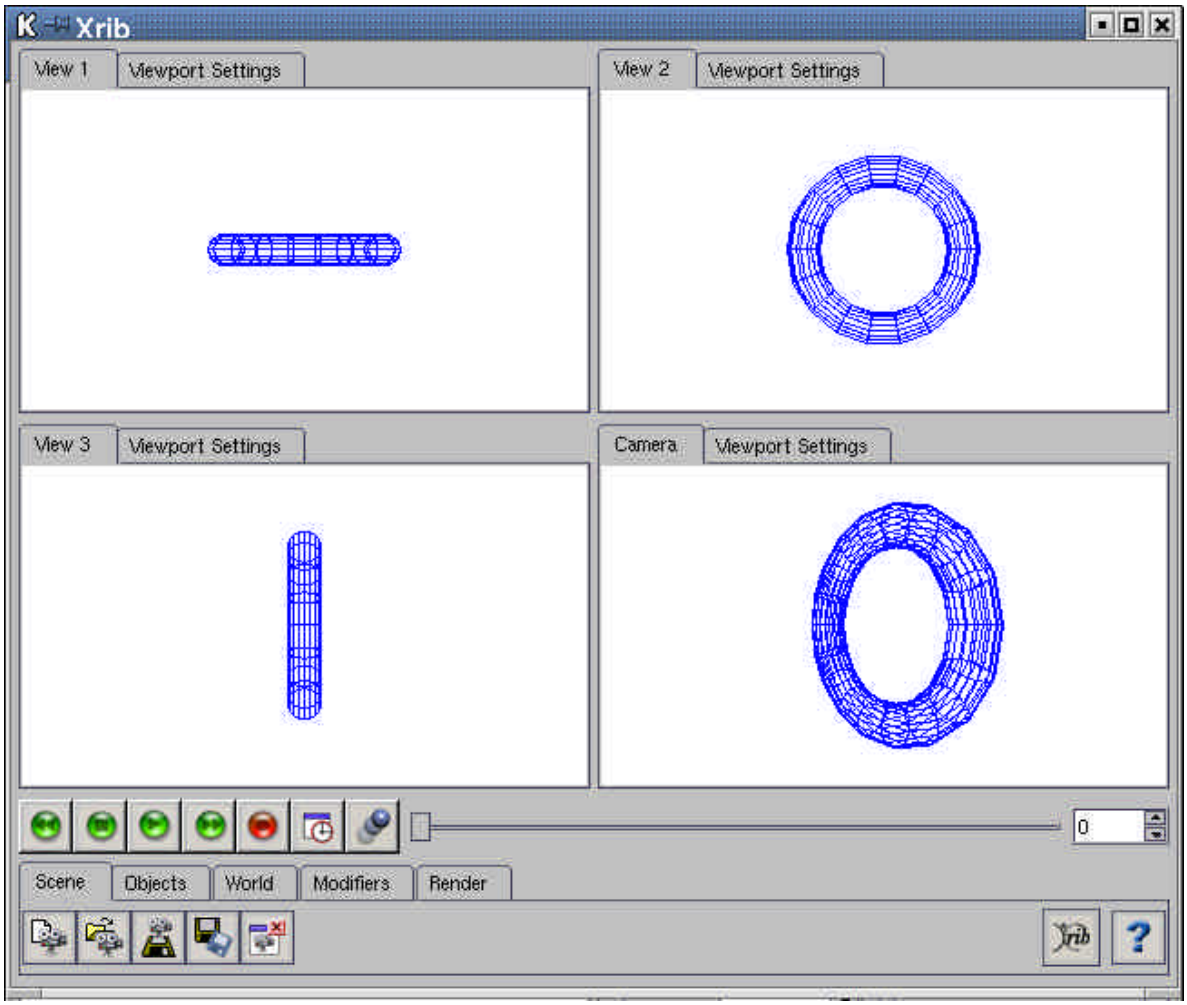


Figura 11. Vista de arranque – Manejo de Archivos

Como se había mencionado, la aplicación se desarrolló en C++ usando las bibliotecas QT y OpenGL y su código se distribuye con licencia GP.

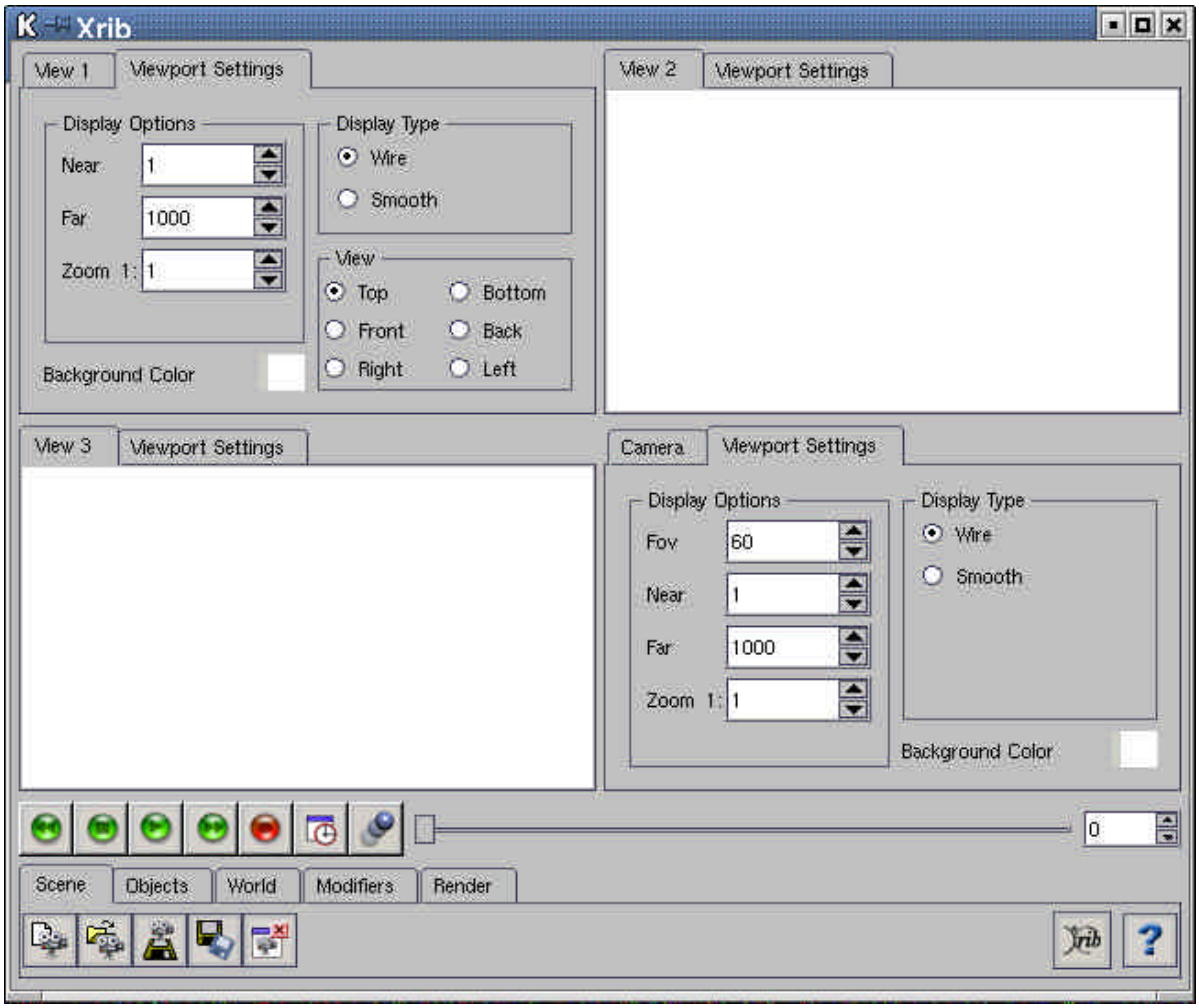


Figura 12. Configuración de las vistas

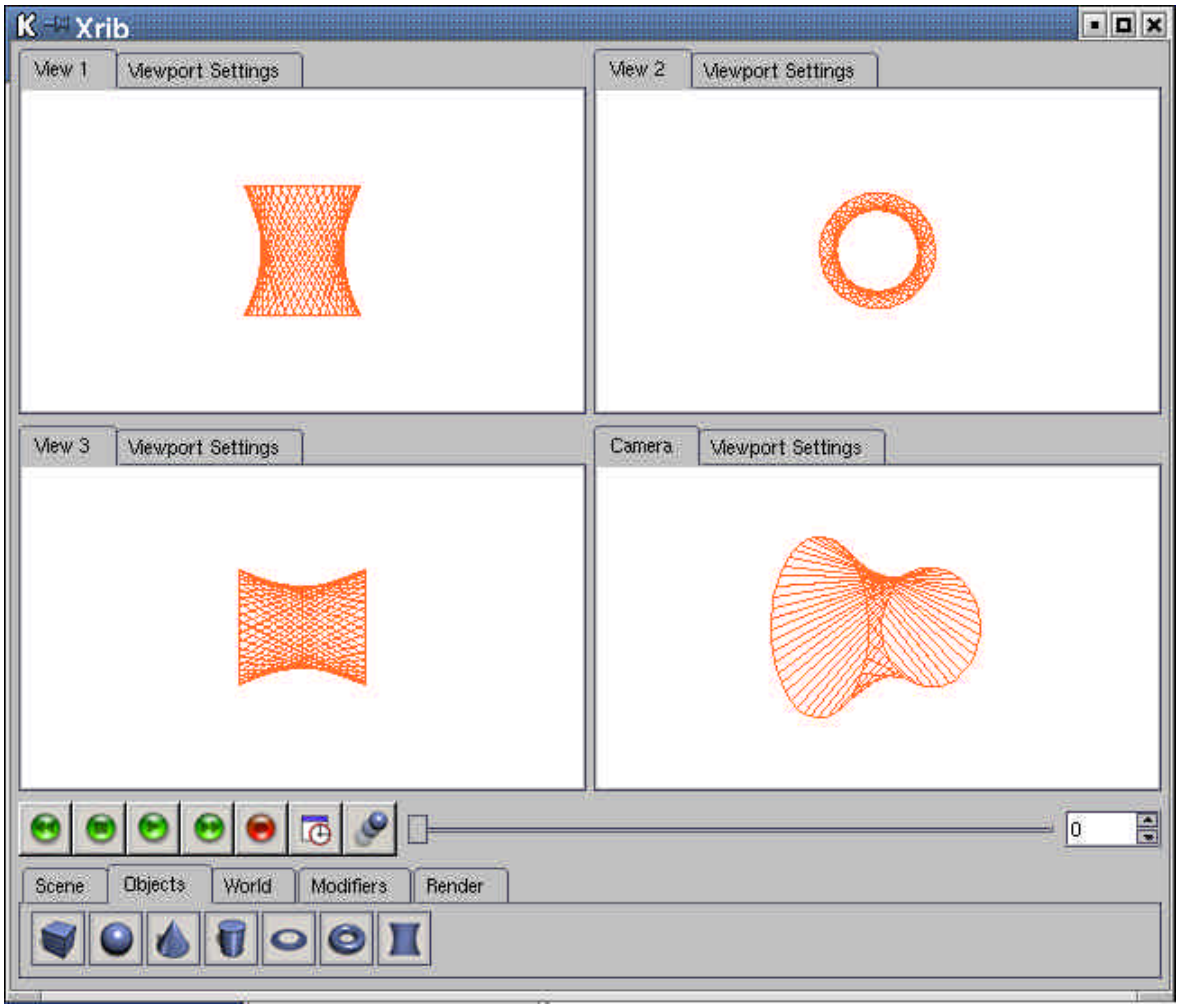


Figura 13. Creación de Objetos.

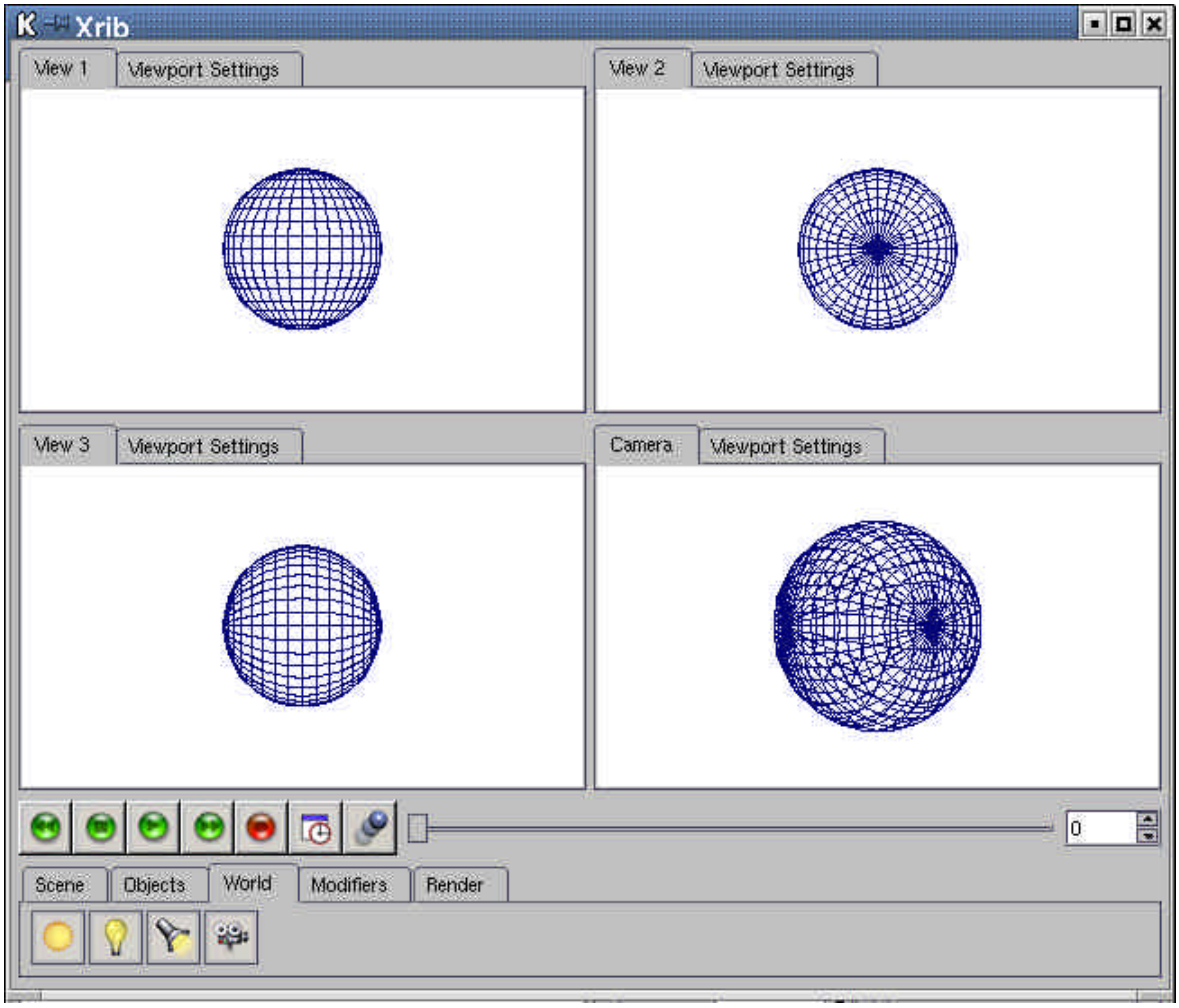


Figura 14. Manejo del Mundo (Luces y Cámara)

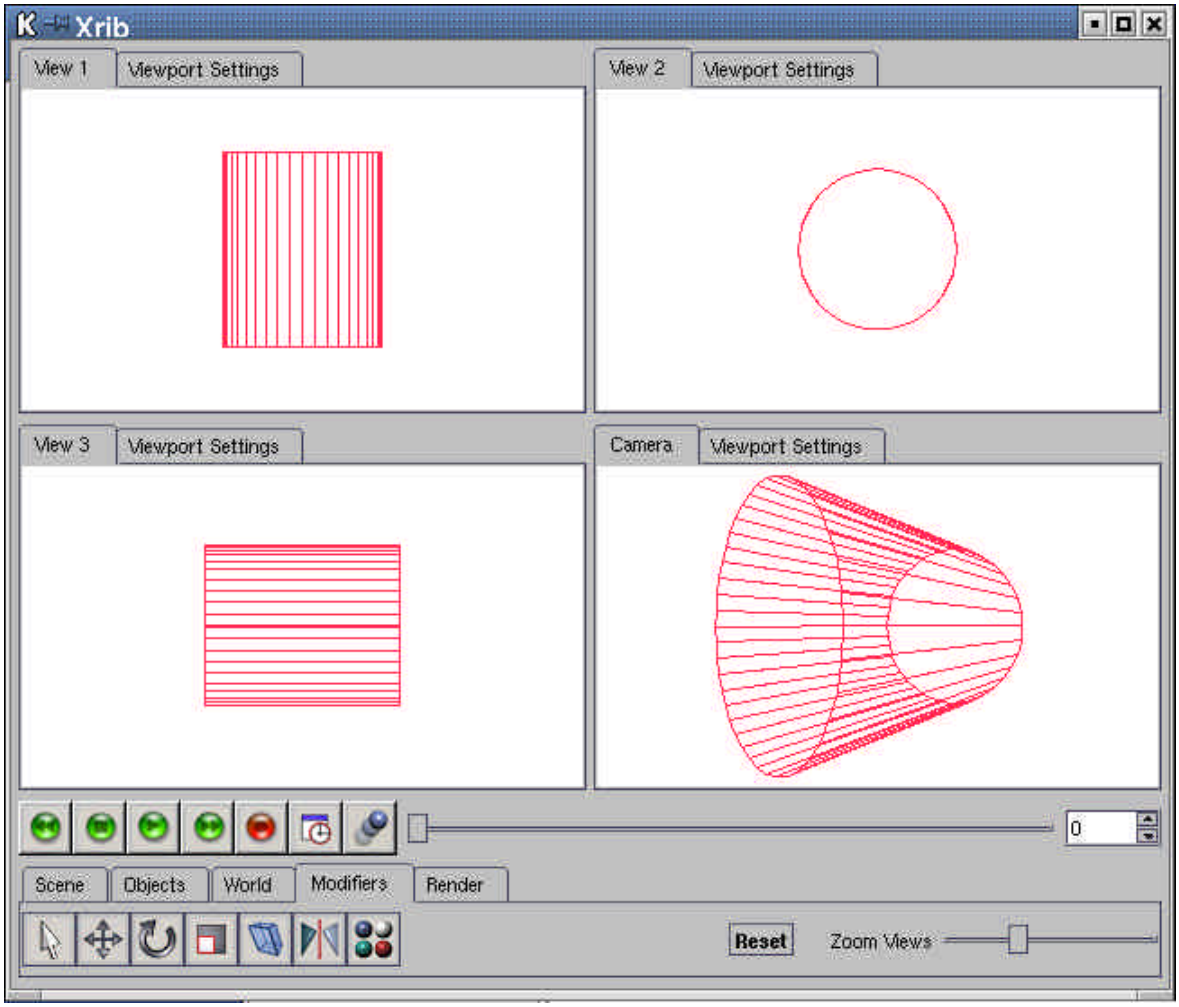


Figura 15. Modificadores de primitivas

(Transformaciones y Material).

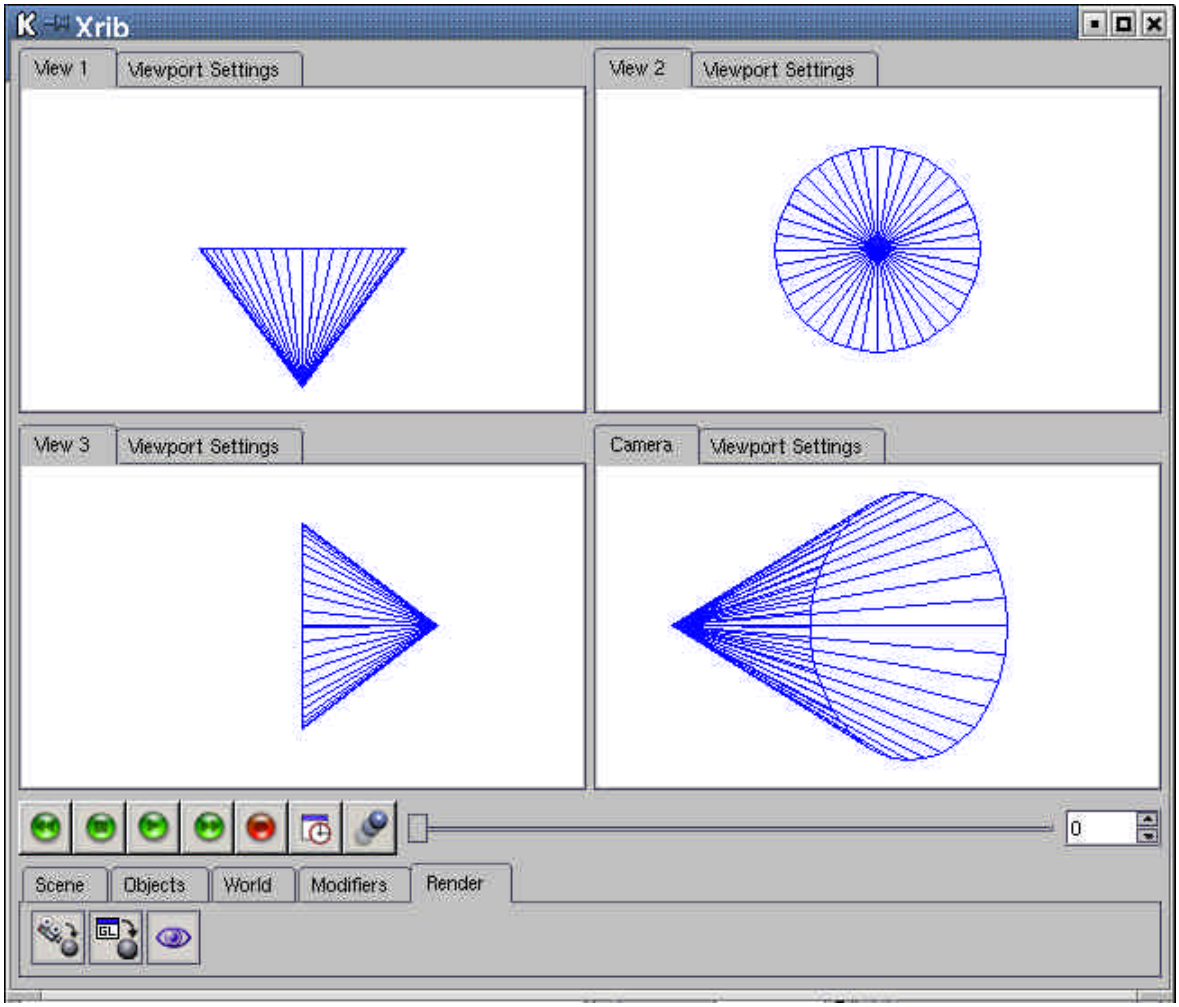


Figura 16. Renderizar Escena.

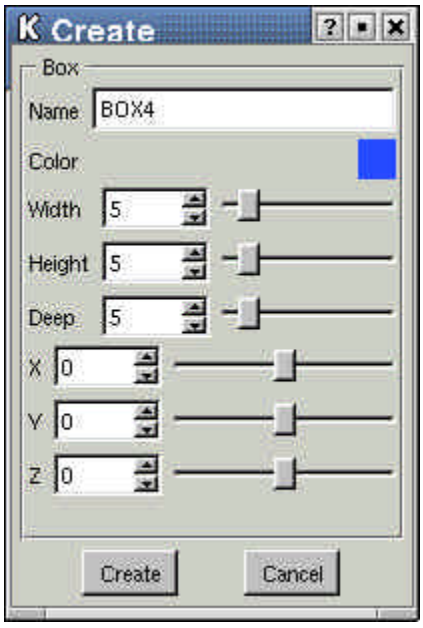


Figura 17. Creación de Caja

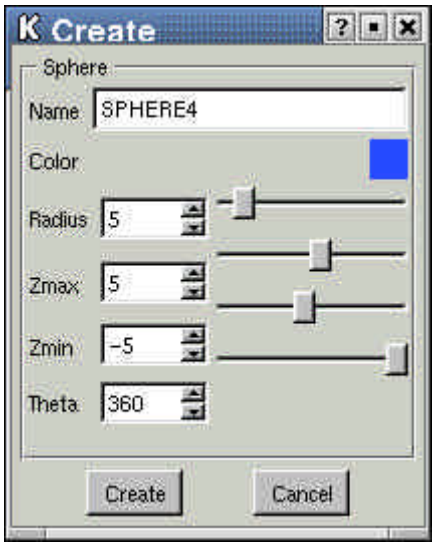


Figura 18. Creación de Esfera

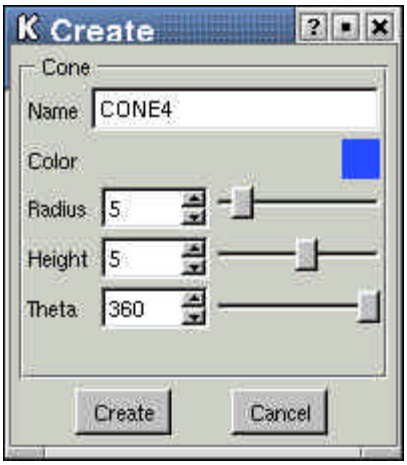


Figura 19. Creación de Cono

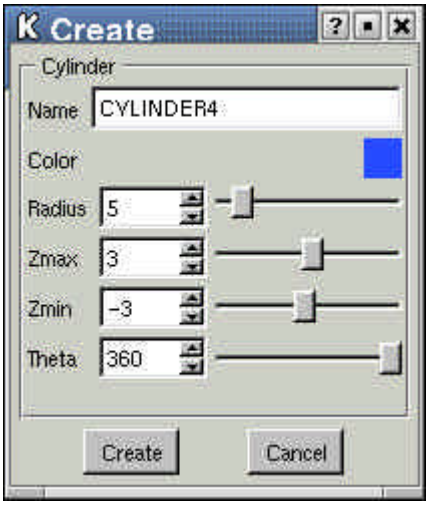


Figura 20. Creación de Cilindro

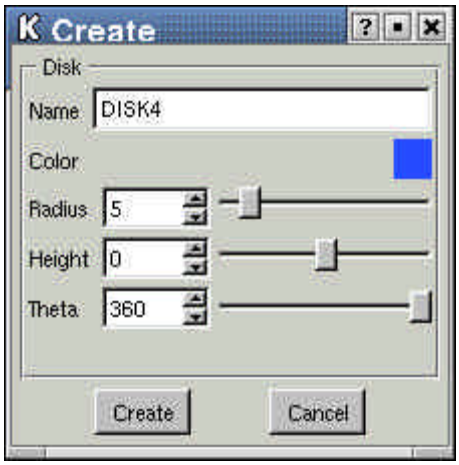


Figura 21. Creación de Disco

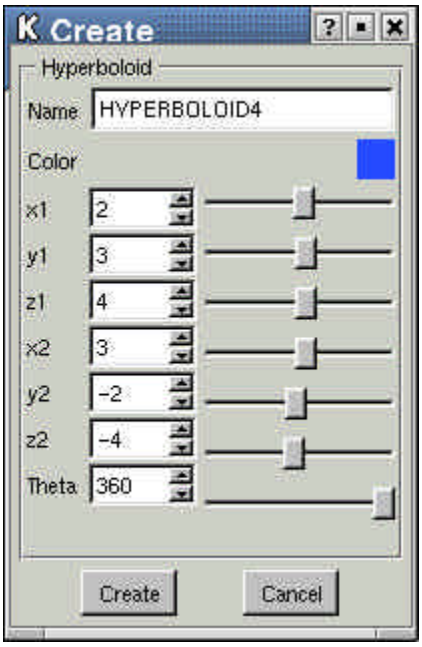


Figura 23. Creación de Hiperboloide

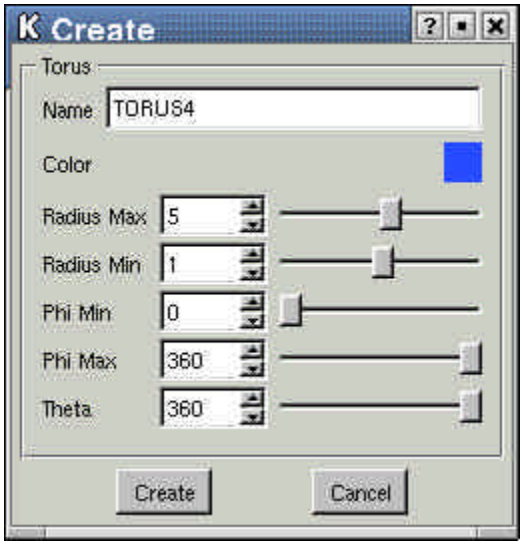


Figura 22. Creación de Toro

Figura 24. Creación de Luz Ambiental

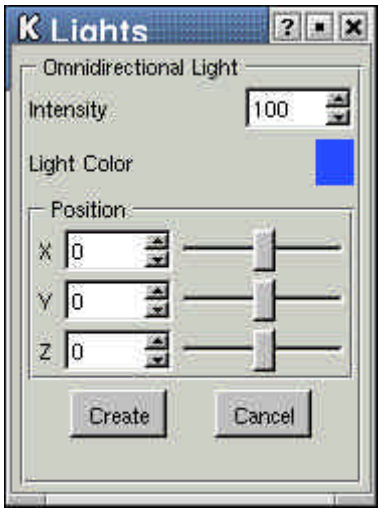


Figura 25. Creación de Luz Onmidireccional

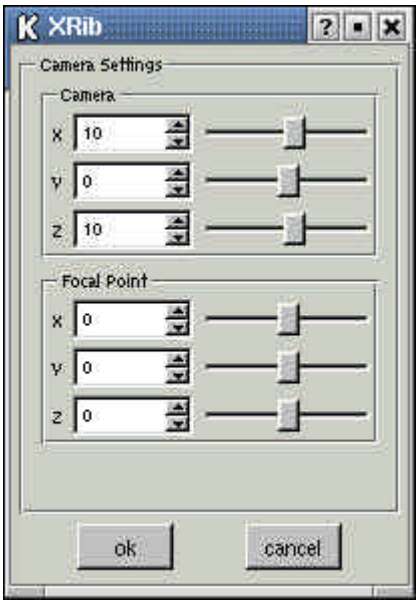


Figura 27. Modificación de Cámara

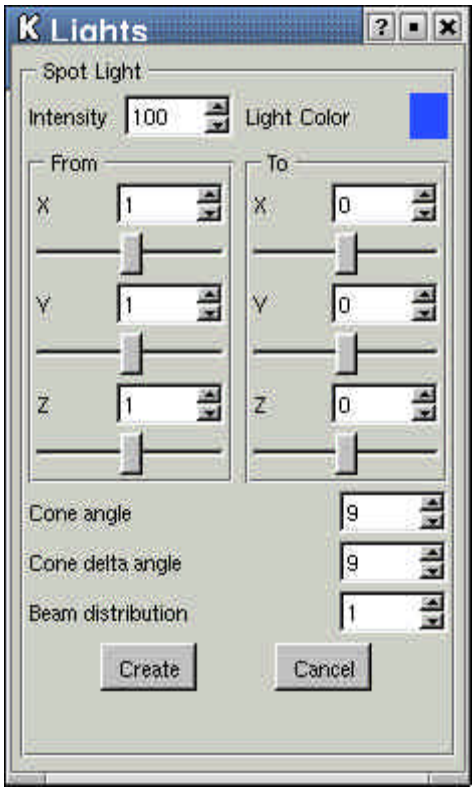


Figura 26. Creación de Luz Direccional

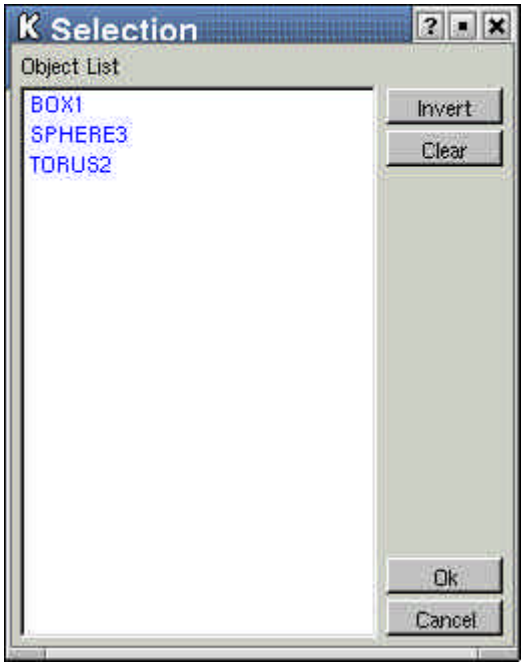


Figura 28. Selección de Objetos

Figura 29. Transformación: Trasladar

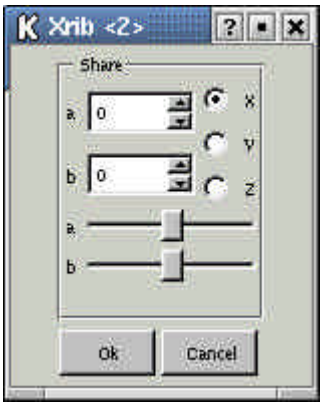


Figura 32. Transformación: Recortar

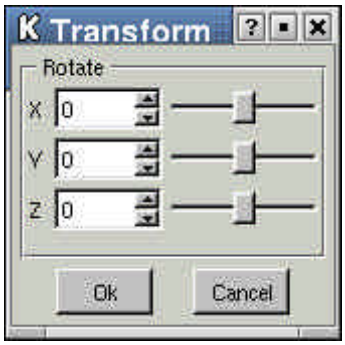


Figura 30. Transformación: Rotar

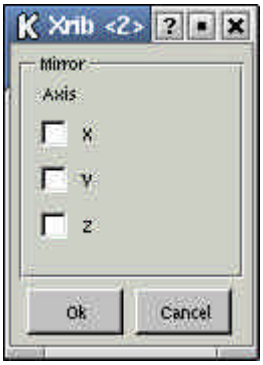


Figura 33. Transformación: Reflejar

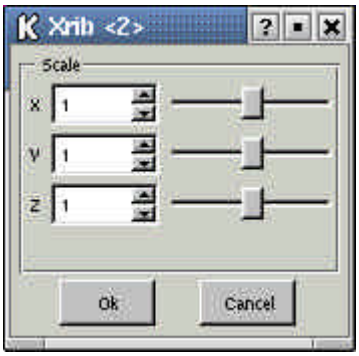


Figura 31. Transformación: Escalar

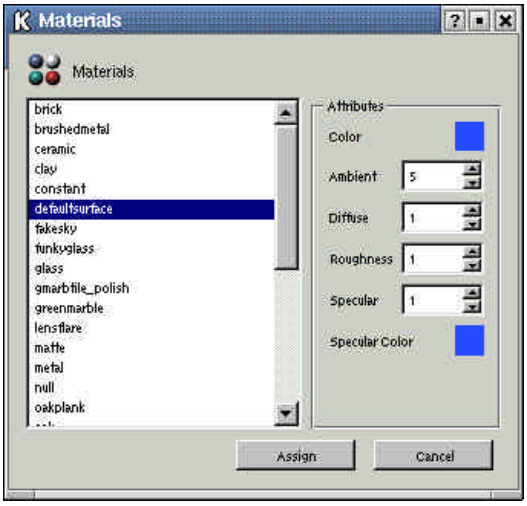


Figura 34. Asignar Materiales 1

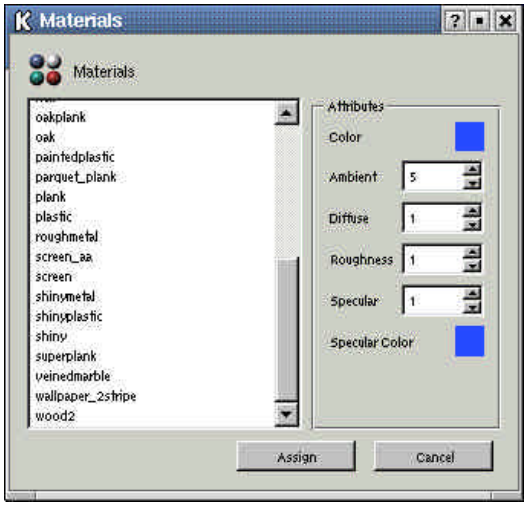


Figura 35. Asignar Materiales 2

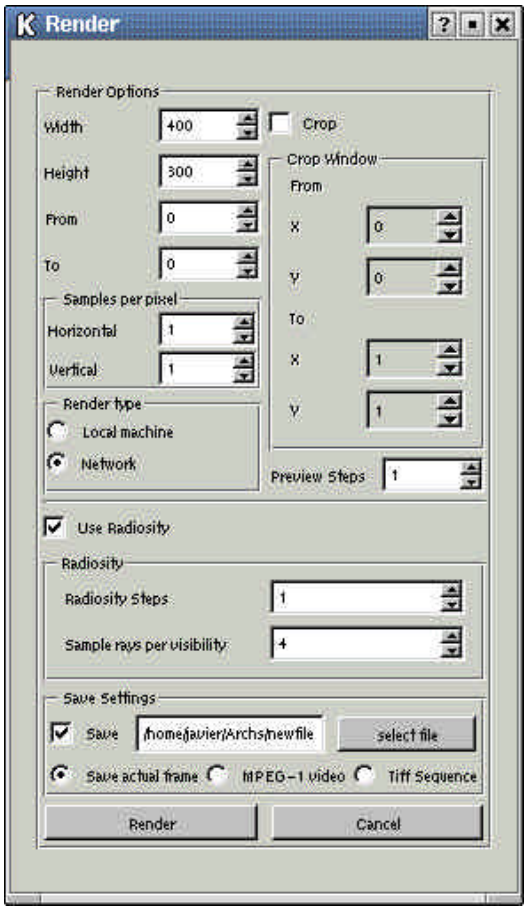


Figura 36. Render

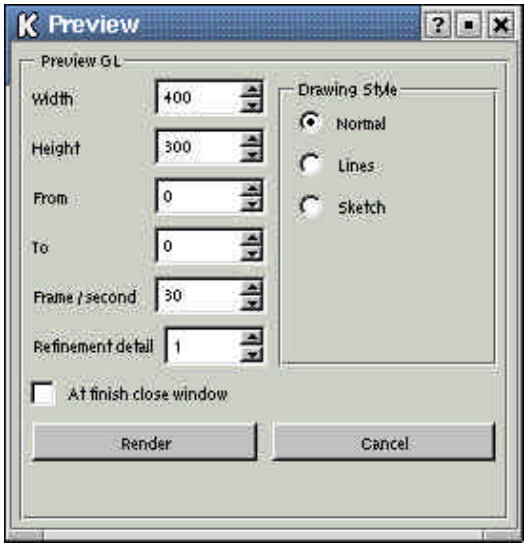


Figura 37. Previsualizar Escena

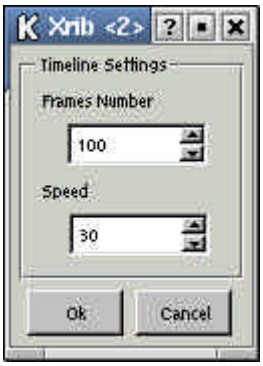


Figura 38. Modificar Timeline

8. CONCLUSIONES

- Se desarrolló una herramienta de software para la visualización, modelización y animación de escenas tiridimensionales usando el enfoque orientado a objetos y la notación UML y se implementó en C ++ haciendo uso del API QT bajo la plataforma linux.

- La herramienta de software desarrollada ¨Xrib¨ ha cumplido a carta cabal con los objetivos planteados además de satisfacer en su totalidad los requerimientos propuestos durante la fase de análisis.

- Se realizó un estudio a fondo de la interfaz Renderman 3.2 de Pixar y de la biblioteca gráfica OpenGL, estudio con el cual se sentaron las bases para el desarrollo de la herramienta de software Xrib

- Con la realización de este proyecto se ha contribuido al mundo de la animación por computadora ya que se ha proveído de una herramienta bajo la licencia GPL, la cual permite su futuro desarrollo como base de una herramienta profesional para la animación y modelado de escenas 3D.

- El algoritmo de rendering de BMRT provee un soporte casi completo (excepto distorsión de movimiento y profundidad de campo) a las escenas definidas con la interfaz Renderman, característica que nos permitió usar las ventajas de Renderman como interfaz para la definición de escenas tridimensionales.

- La alta calidad visual de las imágenes generadas por rendrib y el formato tiff de 32 bits con canal alpha al cual se pueden almacenar las imágenes permiten la utilización de Xrib como la herramienta perfecta para generar secuencias de imágenes en tres dimensiones bajo linux y posteriormente realizar vídeos con sonidos y mezclas de capas de vídeo en herramientas de postproducción de video digital.

9. RECOMENDACIONES

- Una de las características más buscadas por los usuarios de software de para la realización de escenas tridimensionales es la de interpolación entre cuadros claves, es decir el cálculo de las posiciones intermedias de los elementos de la escena entre dos cuadros claves; por lo que se recomienda para futuras implementaciones la adición de esta característica.
- Se recomienda para futuras implementaciones la realización del módulo de carga de escenas definidas en formato rib, y no solamente xrib como actualmente se hace.

10. BIBLIOGRAFÍA

LARMAN Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. Editorial Prentice Hall. 1999

CHOVER SELLÉS Miguel. Prácticas de Informática Gráfica. Publicaciones de la Universitat Jaume I. 1998

HEARN Donald, BAKER M.Pauline. Gráficas por Computadora. Editorial Prentice Hall. 1994

PIXAR. The RenderMan Interface Version 3.2, 2000.

BMRT- Blue Moon Rendering Tools Page. Disponible en: http://www.bmrt.org, 2000.

UPSTILL Steve. The RenderMan Companion. Addison Wesley. 1990.