Cálculo de intersecciones rayo/objeto en ray tracing: Una especificación formal en Z

Computation of Ray / Object Intersections in Ray Tracing: A Formal Specification in Z

Ernesto Rivera-Alvarado
Computer Science
Costa Rica Institute of Technology Costa Rica
ernestoriv7@yahoo.com

Kervin Sánchez-Herrera
Computer Science
Costa Rica Institute of Technology Costa Rica
krsnhr@gmail.com

Ignacio Trejos-Zelaya
Computer Science
Costa Rica Institute of Technology Costa Rica
itrejos@tec.ac.cr

Fecha de recibido: 28 de febrero 2020

Fecha de aprobado: 11 de abril de 2020

Resumen—Este trabajo presenta un caso de estudio en la especificación formal, mediante la notación Z, del cálculo de intersecciones rayo/objeto en el algoritmo de ray tracing. En el documento se describen los conceptos de escena, objetos geométricos tridimensionales, cálculo de intersecciones rayo/objeto mediante conjunto de puntos y la identificación de la primera intersección del rayo con un objeto de la escena. Dicha información es indispensable para la síntesis de una imagen tridimensional por medio de ray tracing. El caso de estudio es parte de una iniciativa de investigación y educación en la aplicación de métodos formales de descripción y verificación a distintas áreas temáticas dentro de la Computación. Nuestra especificación muestra un alto nivel de abstracción en la descripción de las intersecciones rayo/objeto en problemas de trazado de rayos, un problema importante en Computación Gráfica. Esta exploración invita a describir otros problemas relacionados con Gráficas por computadoras y también abre oportunidades para la verificación formal de las propiedades descritas en las especificación, el desarrollo de algoritmos verificables contra la especificación formal, o el refinamiento de programas correctos por construcción mediante la

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

transformaciones sucesivas que partan de la especificación abstracta.

Palabras clave—Ray tracing, Notación Z, especificación formal, intersección rayo/objeto, primitivas matemáticas, escenas tridimensionales, primera intersección.

Abstract — This work presents a case study in the formal specification, using Z notation, of the calculation of ray / object intersections in the ray tracing algorithm. The document describes the concepts of scene, three-dimensional geometric objects, calculation of ray / object intersections by means of a set of points and the identification of the first intersection of the ray with an object in the scene. This information is essential for the synthesis of a three-dimensional image by means of ray tracing. The case study is part of a research and educational initiative in the application of formal description and verification methods to different subject areas within Computing. Our specification shows a high level of abstraction in the description of ray / object intersections in ray tracing problems, a major problem in Computer Graphics. This exploration invites to describe other problems related to Computer Graphics and also opens opportunities for the formal verification of the properties described in the specifications, the development of verifiable algorithms against the formal specification, or the refinement of correct programs by construction through successive transformations. starting from the abstract specification.

Keywords — Ray tracing, Z notation, formal specification, ray / object intersection, math primitives, three-dimensional scenes, first intersection.

I. INTRODUCCIÓN

El nivel de realismo de las imágenes generadas a través del algoritmo de ray tracing es superior a las generadas por otras técnicas tales como ray casting o rasterización [18, 20]. Sin embargo, este alto nivel de realismo se paga con tiempo de sintetizado por lo que la técnica de trazado de rayos no es viable para aplicaciones que requieren de la generación de imágenes en tiempo real, tales como las que encontramos en videojuegos, animación y realidad virtual. Según [16], la mayor parte del tiempo de ejecución del algoritmo de ray tracing se utiliza en el cálculo de la intersección de los rayos generados por el algoritmo con los objetos que conforman la escena. En un ray tracer

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

básico esta operación se ejecuta en un tiempo $I * N$, donde I corresponde a la cantidad de pixeles que conforman la imagen y N corresponde a la cantidad de objetos que forman parte de la escena.

El ray tracing y los mecanismos para detectar colisiones han sido ampliamente estudiados e implementados en distintos lenguajes de programación [8], sin embargo, no encontramos en la literatura una especificación formal del cálculo de intersecciones rayo/objeto, ni de sus sub-componentes. Este documento presenta un primer caso de estudio de especificación formal, mediante la notación Z, el cálculo de la intersección rayo/objeto en un algoritmo de ray tracing.

Figura 1. ((Der Zeichner des liegenden Weibes)), Durero, 1525

II. MARCO CONCEPTUAL

En esta sección se discute sobre los elementos fundamentales que componen al algoritmo de ray tracing con el fin de comprender de mejor manera el trabajo desarrollado en el presente documento.

II-A. ¿Cómo funciona el algoritmo de ray tracing?

Como su nombre en inglés lo expresa, ray tracing funciona mediante la generación de rayos matemáticos desde un origen hacia cada uno de los pixeles que se despliegan en la pantalla, todo esto dentro del contexto de un mundo tridimensional que contiene distintos objetos descritos en una escena. Para cada rayo se detecta y calcula la intersección con los objetos que forman parte de la escena, y se reporta el color del objeto más cercano con el fin de que este sea pintado en la pantalla.

Para lograr esto, una ventana de proyección y un ojo virtual son definidos en un espacio tridimensional. Desde el ojo se generan rayos hacia la ventana de proyección, se detecta la colisión con el objeto más cercano, y el color correspondiente a este es asignado al pixel de la pantalla con las

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

coordenadas que se utilizaron para generar el rayo. La pintura de Durero que se aprecia en la Figura 1 incluye todos los elementos utilizados en ray tracing, en donde el ojo del artista representa al ojo de la escena desde el cual se generan los rayos, el plano en el medio representa el plano de proyección a través del cual viajan los rayos, la dama que posa representa la escena tridimensional y el papel en el cual el artista dibuja representa lo que se despliega en pantalla. En caso de que el rayo trazado no detecte ninguna colisión, un color por defecto es

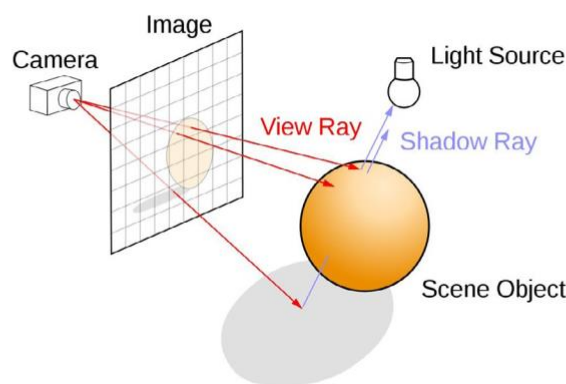

Figura 2. Mecanismo de ray tracing

Fuente: [3]

II-B. El algoritmo de ray tracing

El algoritmo de ray tracing consiste en dos ciclos **for** anidados, en donde el **for** externo recorre los pixeles verticales, mientras que el **for** subordinado recorre los pixeles horizontales. Para cada pixel se genera un rayo. Dicho rayo debe procesarse con todos los objetos geométricos que componen la escena, con el fin de detectar si existe intersección con dichos objetos e identificar el objeto más cercano al origen del rayo para reportar su color en pantalla. El Algoritmo II.1 muestra la composición interna del ray tracer.

Algoritmo II.1: *RAYTRACER (Escena)*

$$\textbf{for } y \leftarrow 1 \textbf{ to } Escena.ResolucionVertical$$
$$\textbf{do} \begin{cases} \textbf{for } x \leftarrow 1 \textbf{ to } Escena.ResolucionHorizontal \\ \textbf{do} \begin{cases} Rayo \leftarrow \text{OBTENERRAYO}(x, y, Escena.ojo) \\ Interseccion \leftarrow \text{INTERSECCION}(Rayo, Escena) \\ \text{PINTARPANTALLA}(x, y, Interseccion) \end{cases} \end{cases}$$

La Figura 2 muestra y ejemplifica cómo se da el recorrido de rayos a través del algoritmo y su respectiva intersección con objetos que componen la escena [16].

II-C. Cálculo de la intersección más cercana

La especificación formal, expresada en notación Z, se realizó únicamente para este sub-componente del algoritmo de ray tracing. Se escogió

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

esta parte específica debido a que es una de las tareas más críticas en el tiempo de ejecución del *ray tracer* y es en la cual se ha concentrado mayormente la investigación científica [16]. La invocación de este componente aparece como INTERSECCION (Rayo, Escena) en el Algoritmo

II.1. Esta función recibe un rayo matemático como entrada y la descripción de una escena compuesta por diferentes formas geométricas, con el fin de identificar las intersecciones del rayo con los objetos de la escena (en que caso de que los haya) y determinar cuál es la intersección que se encuentra a una distancia euclidiana menor con respecto del origen del rayo.

Esta función reporta la intersección más cercana y el objeto correspondiente a esa intersección. El Algoritmo II.2 muestra la composición interna de esta función.

Algoritmo II.2: *INTERSECCION(Rayo; Escena)*

```
Inter ← ∅
for each objeto ∈ Escena
    do { if EXISTEINTERSECCION(Rayo, Objeto)
            then Inter agrega [PUNTO(Rayo, Objeto), Objeto]
if Inter = ∅
    then output (∅)
else output (INTERSECCION(Inter))
```

II-D. Efecto de la iluminación en el color que es pintado en pantalla

Cada escena por ser sintetizada mediante el algoritmo de ray tracing contará con una o más fuentes de iluminación, las cuales tendrán un impacto directo en el color que será reportado para ser pintado en pantalla.

Ley de Lambert corresponde al mecanismo mediante el cual la iluminación impacta el color reportado. Dicha ley indica que la intensidad de la iluminación para un punto dado en un objeto es proporcional al coseno del ángulo entre el vector que va desde el punto de intersección del rayo con el objeto, hacia la fuente de iluminación y el vector normal a la superficie en la cual se localiza dicho punto. Esta

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

relación se puede apreciar en la Ecuación II-D [8].

$$I = (L . N)$$

En dicha ecuación se puede observar que se utiliza el producto punto como una alternativa al uso de la relación cosenoidal directa entre el ángulo y el vector. La intensidad I corresponde a un número entre 0 y 1. Dicho número multiplica a cada uno de los componentes RGB del color del objeto. Después de realizar esta operación, el color resultante se reporta para ser desplegado en pantalla [8]. La Figura 2 también muestra el vector que va desde el punto de intersección del rayo con el objeto hacia la fuente de iluminación. Dado que la intensidad varía en función de la ubicación del punto en donde el rayo interseca al objeto, al reportar el color haciendo uso de esta técnica se obtiene un efecto de tridimensionalidad muy realista[16]. Esto se puede apreciar en la Figura 3, en la cual una esfera fue sintetizada haciendo uso de *ray tracing*.

II-E. Sombras

En el algoritmo de ray tracing determinar si un punto en la escena contiene una sombra es bastante sencillo. Esto se logra verificando si hay un camino sin obstáculos desde el punto que se está evaluando hacia la fuente de iluminación.

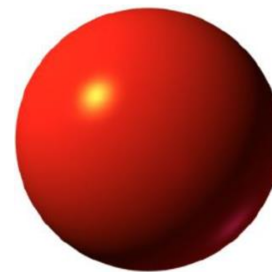

Figura 3. Esfera sintetizada haciendo uso de ray tracing y la Ley de Lambert
Fuente: [9]

Esto se puede verificar haciendo uso del mismo algoritmo de *ray tracing* con la diferencia de que se asigna como origen el punto de intersección del rayo original con dirección hacia la fuente de luz, por lo tanto si se detecta una intersección con un objeto, debe de reportarse una sombra[8]. Para la asignación de sombras en un punto de la escena se debe de recurrir al cálculo de la intersección rayo/objeto, con la diferencia que se debe detectar si hubo intersección con algún objeto, sin tener que reportar el objeto más

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

cercano contra el cual se dio la intersección.

II-F. Elementos necesarios para la descripción de una escena

A continuación se listan los elementos necesarios para describir una escena que será sintetizada a través del algoritmo de ray tracing.

1. Ojo/cámara: Este es el punto de inicio desde donde se originarán los rayos hacia la escena tridimensional.
2. Resolución: Se define en pixeles y delimita el tamaño máximo de la imagen así como el recuadro a través del cual se trazarán los rayos hacia la escena.
3. Objetos que componen la escena: Estos serán figuras geométricas definidas en un espacio de tres dimensiones. Las figuras deben ser definidas paramétricamente para poder calcular su intersección con el rayo. Las figuras geométricas que se especificarán formalmente en este documento son estas:
   - Esferas.
   - Cilindros.
   - Conos.
4. Fuentes de iluminación: Estas determinan los efectos de iluminación y tridimensionalidad que se apreciarán en la escena en función de su posición y de la posición del ojo/cámara.

II-G. Trabajo relacionado

Múltiples autores han trabajado en la investigación, implementación y desarrollo del algoritmo de *ray tracing* en distintos lenguajes de programación [5, 8, 16, 25], sin embargo, ninguna de las referencias brinda una especificación formal, sino que presentan explicaciones y descripciones específicas para los lenguajes de programación en los cuales se implementó el algoritmo (C, C++, Python, entre otros).

Otros investigadores han utilizado notaciones agnósticas a los lenguajes de programación (pseudocódigo) para optimizar componentes internos del algoritmo de *ray tracing*, como por ejemplo las estructuras de aceleración de objetos de la escena

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

[22, 23], o su compresión para mejorar los tiempos de transferencia de datos en memoria [27]. Estos trabajos resuelven problemas importantes de rendimiento, sin embargo, carecen de una descripción formal de los mecanismos utilizados. Otros enfoques han utilizado lenguajes de descripción de hardware para la implementación del algoritmo en circuitos electrónicos programables (FPGAs) [2]. La diferencia de esta clase de lenguajes en comparación a los lenguajes de programación tradicionales es que éstos describen un circuito electrónico que realiza una función digital, por lo que se deben mapear todos los elementos y componentes del algoritmo de *ray tracing* a funciones binarias que realizan las operaciones necesarias para sintetizar una imagen. Otros esfuerzos específicos han descrito las funcionalidades del algoritmo de ray tracing en el nivel matemático, sin hacer uso de una notación formal [21]. Por último, distintos enfoques de trabajos se han concentrado en la descripción de implementaciones de ray tracing en hardware específico [12, 13, 24], pero nuevamente usan notaciones ligadas a los lenguajes de programación utilizados, pseudocódigo o de descripción de hardware, sin hacer uso de una especificación formal.

El modelaje preciso de las propiedades de los sistemas son algunos de los beneficios que aporta el uso de los lenguajes de especificación formales [14]. Tales lenguajes ofrecen estructuras matemáticas, abstractas y precisas, para modelar y describir sistemas informáticos, como los relacionados con gráficas computacionales. Su fundamento lógico-matemático abre oportunidades para demostrar propiedades de las descripciones, así como para la obtención de programas correctos por construcción o demostrablemente correctos (vía inferencias lógico-matemáticas) [4].

La notación Z está basada en la teoría de conjuntos con tipos y la lógica matemática [19], [26]. Z es un lenguaje de especificación formal propuesto por Jean- Raymond Abrial [1] y desarrollado en la Universidad de Oxford durante la década de 1980 [15]. En Z, los objetos matemáticos y

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

sus propiedades restringidas por predicados pueden reunirse en esquemas (schemas) [26], lo que facilita la abstracción y la documentación de documentos estructurados. Z tiene una sintaxis formal y un estricto sistema de tipos (todo objeto tiene un tipo), lo cual excluye muchas inconsistencias en los documentos de descripción. Hay un estándar ISO para Z [10], así como un Manual de Referencia que es el estándar de facto, escrito por Spivey [19]. Existen herramientas para validar sintaxis y tipos de los documentos escritos en Z, así como otras para hacer razonamiento orientado a la verificación de propiedades [17].

Aunque el campo de la graficación computacional tiene un fuerte sustento matemático, es muy escasa la especificación formal del dominio y sus problemas de interés. En 1983, Carson [6] desarrolló un marco conceptual para especificar formalmente sistemas de graficación computacional, para apoyar el desarrollo de estándares de computación gráfica independiente de los dispositivos. Duce desarrolló descripciones formales como parte del proceso del desarrollo del estándar de ISO para programación de gráficos computacionales (Graphical Kernel System (GKS) [7]. Dentro de la literatura de Z, solamente en el libro de Jacky [11] se desarrolla un corto caso de estudio sobre Gráfica y Geometría computacionales.

III. ESPECIFICACIÓN FORMAL DE RAY TRACING EN Z

En esta sección se brinda la especificación formal de los tipos de datos, escena, funciones auxiliares y del cálculo de las intersecciones rayo/objeto.

III-A. Tipos de datos y constantes globales Se define el tipo de los números reales que son necesarios para el posicionamiento de objetos en la escena y en el uso de variables continuas.

$$[\mathbb{R}]$$

De la misma manera, se define el tipo dado por los números reales positivos. Estos son necesarios para representar cantidades estrictamente positivas, como las distancias.

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

$$Rmas == \{rmas: \mathbb{R} | rmas > 0\}$$

En el siguiente esquema se define el tipo **Punto**, el cual está compuesto por tres componentes del tipo de los números reales. Estos se utilizan para representar posiciones dentro del espacio tridimensional de la escena.

Punto
$x, y, z : \mathbb{R}$

El tipo **Vector**, al igual que el tipo Punto, consta de tres componentes definidas como números reales. Como en la matemática clásica, el tipo Vector se utiliza para denotar direcciones.

Vector
$x, y, z : \mathbb{R}$

El rayo matemático está compuesto por un Punto que denota el origen del rayo (en el contexto de *ray tracing* este corresponde a la ubicación del ojo en la escena) y un *Vector*, el cual indica la dirección hacia la cual viaja el rayo. Dicha dirección varía dependiendo del pixel del cual que se está calculando el color por desplegar en pantalla.

Rayo
$p : Punto$
$v : Vector$

Se definen dos conjuntos: el conjunto *Elemento* corresponde a todos los elementos que pueden formar parte de una escena. El conjunto *Puntos* corresponde al conjunto de todos los puntos del tipo Punto que puede encontrarse en una escena.

$$[Elemento, Puntos]$$

En la presente especificación está limitado a escenas que contienen únicamente esferas, conos y cilindros. Esto implica que los elementos que componen una escena podrán ser únicamente de los tres tipos que se especifican a continuación:

$$Tipo ::= soy_esfera \mid soy_cilindro \mid soy_cono$$

Como se observará más adelante, se especificaron los esquemas para la construcción de una escena. Los esquemas que agregan y eliminan elementos de una escena brindan una respuesta específica. Las posibles respuestas que pueden brindar dichos esquemas se describen a continuación.

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

$$
\begin{aligned}
\mathit{Respuesta} ::= \;& \mathit{elemento_agregado} \\
& \mid \mathit{elemento_eliminado} \\
& \mid \mathit{esfera_agregada} \\
& \mid \mathit{cono_agregado} \\
& \mid \mathit{cilindro_agregado} \\
& \mid \mathit{esfera_eliminada} \\
& \mid \mathit{cono_eliminado} \\
& \mid \mathit{cilindro_eliminado} \\
& \mid \mathit{puntos_agregados}
\end{aligned}
$$

III-B. Definición de Escena

Definimos una escena como el conjunto de elementos que la conforman, en donde cada elemento tiene un tipo y este tipo es único. Puesto que hay varias clases diferentes de elementos que componen una escena. Estos tienen características específicas para cada clase, la escena contiene funciones que permiten obtener las características propias de un elemento que forma parte de la escena. Las características específicas por cada elemento que forma parte de la escena son las siguientes:

- Esfera: centro (tipo *Punto*) y radio (tipo *Rmas*).
- Cilindro: centro (tipo *Punto*), radio (tipo *Rmas*), orientación (tipo *Vector*) y altura (tipo *Rmas*).
- Cono: centro (tipo *Punto*), proporción (relación del radio por cada unidad de altura (tipo *Rmas*), orientación (tipo *Vector*) y altura (tipo *Rmas*).

El esquema Escena contiene una función que, dado un elemento, permite determinar si es esfera, cono o cilindro.

$$
\begin{array}{l}
\textbf{Escena} \\
\hline
\mathit{elementos} : \mathbb{P}\,\mathit{Elemento} \\
\mathit{tipos} : \mathbb{P}\,\mathit{Tipo} \\
\mathit{de_que_tipo} : \mathit{Elemento} \nrightarrow \mathit{Tipo} \\
\mathit{mi_radio} : \mathit{Elemento} \nrightarrow \mathit{Rmas} \\
\mathit{mi_centro} : \mathit{Elemento} \nrightarrow \mathit{Punto} \\
\mathit{mi_vector} : \mathit{Elemento} \nrightarrow \mathit{Vector} \\
\mathit{mi_proporcion} : \mathit{Elemento} \nrightarrow \mathit{Rmas} \\
\mathit{mi_altura} : \mathit{Elemento} \nrightarrow \mathit{Rmas} \\
\hline
\mathit{elementos} = \mathrm{dom}\,\mathit{de_que_tipo} \\
\mathit{elementos} = \mathrm{dom}\,\mathit{mi_centro} \\
\mathrm{dom}\,\mathit{mi_radio} \subseteq \mathit{elementos} \\
\mathrm{dom}\,\mathit{mi_vector} \subseteq \mathit{elementos} \\
\mathit{tipos} = \mathrm{ran}\,\mathit{de_que_tipo} \\
\hline
\end{array}
$$

Seguidamente definimos los esquemas que permiten agregar y eliminar elementos a la escena. El esquema *Agregar_Elemento* es general y no contiene la información necesaria para agregar un elemento específico. Las esquemas para agregar y eliminar elementos específicos tales como esferas, conos

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

y cilindros, se construirán de manera incremental a partir de los dos esquemas que se definen a continuación

$$
\begin{array}{l}
\textit{Agregar_Elemento} \\
\hline
\Delta Escena \\
e? : Elemento \\
t? : Tipo \\
r! : Respuesta \\
\hline
e? \notin elementos \\
elementos' = elementos \cup \{e?\} \\
de_que_tipo' = de_que_tipo \cup \{e? \mapsto t?\} \\
r! = elemento_agregado \\
\hline
\end{array}
$$

La operación *Eliminar_Elemento* hace lo opuesto a *Agregar_Elemento.*

$$
\begin{array}{l}
\textit{Eliminar_Elemento} \\
\hline
\Delta Escena \\
e? : Elemento \\
r! : Respuesta \\
\hline
e? \in elementos \\
elementos' = elementos \setminus \{e?\} \\
de_que_tipo' = \{e?\} \lhd de_que_tipo \\
r! = elemento_eliminado \\
\hline
\end{array}
$$

A partir de los esquemas anteriores, se definen de manera incremental los esquemas que permiten agregar y eliminar elementos de la escena, los cuales, como fue indicado anteriormente, sólo pueden corresponder al tipo esfera, cilindro y cono.

Los esquemas para agregar y eliminar esferas de la escena se definen a continuación.

$$
\begin{array}{l}
\textit{Agregar_Esfera} \\
\hline
\Delta Agregar_Elemento \\
centro? : Punto \\
radio? : Rmas \\
\hline
t? = soy_esfera \\
mi_centro' = mi_centro \cup \{e? \mapsto centro?\} \\
mi_radio' = mi_radio \cup \{e? \mapsto radio?\} \\
r! = esfera_agregada \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{Eliminar_Esfera} \\
\hline
\Delta Eliminar_Elemento \\
\hline
mi_centro' = \{e?\} \lhd mi_centro \\
mi_radio' = \{e?\} \lhd mi_radio \\
r! = esfera_eliminada \\
\hline
\end{array}
$$

Los esquemas para agregar y eliminar cilindros de la escena se definen a continuación.

$$
\begin{array}{l}
\textit{Agregar_Cilindro} \\
\hline
\Delta Agregar_Elemento \\
centro? : Punto \\
radio? : Rmas \\
vector? : Vector \\
altura? : Rmas \\
\hline
t? = soy_cilindro \\
mi_centro' = mi_centro \cup \{e? \mapsto centro?\} \\
mi_radio' = mi_radio \cup \{e? \mapsto radio?\} \\
mi_vector' = mi_vector \cup \{e? \mapsto vector?\} \\
mi_altura' = mi_altura \cup \{e? \mapsto altura?\} \\
r! = cilindro_agregado \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{Eliminar_Cilindro} \\
\hline
\Delta Eliminar_Elemento \\
\hline
mi_centro' = \{e?\} \lhd mi_centro \\
mi_radio' = \{e?\} \lhd mi_radio \\
mi_vector' = \{e?\} \lhd mi_vector \\
mi_altura' = \{e?\} \lhd mi_altura \\
r! = cilindro_eliminado \\
\hline
\end{array}
$$

Los esquemas para agregar y eliminar conos de la escena se definen a continuación.

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

$$
\begin{array}{l}
\textit{Agregar_Cono} \\
\Delta Agregar_Elemento \\
centro? : Punto \\
proporcion? : Rmas \\
vector? : Vector \\
altura? : Rmas \\
\hline
t? = soy_cono \\
mi_centro' = mi_centro \cup \{e? \mapsto centro?\} \\
mi_proporcion' = mi_proporcion \cup \{e? \mapsto proporcion?\} \\
mi_vector' = mi_vector \cup \{e? \mapsto vector?\} \\
mi_altura' = mi_altura \cup \{e? \mapsto altura?\} \\
r! = cono_agregado
\end{array}
$$

$$
\begin{array}{l}
\textit{Eliminar_Cono} \\
\Delta Eliminar_Elemento \\
\hline
mi_centro' = \{e?\} \lhd mi_centro \\
mi_vector' = \{e?\} \lhd mi_vector \\
mi_altura' = \{e?\} \lhd mi_altura \\
mi_proporcion' = \{e?\} \lhd mi_proporcion \\
r! = cono_eliminado
\end{array}
$$

II-C. Funciones auxiliares y cálculo de la intersección rayo/objeto

Una escena compuesta a partir de elementos geométricos puede ser transformada en una que contiene un conjunto de puntos en un espacio tridimensional. De esta manera se procede a convertir todos los elementos geométricos que componen la escena a un conjunto de puntos. Buscamos describir la escena tridimensional como un conjunto de puntos. Análogamente, el rayo matemático, con el cual se calculará la intersección con los elementos de la escena, se define como un conjunto de puntos. Por último también se incorpora una función que, dado un punto, determina a cuál elemento corresponde.

Se define la función auxiliar *cuadrado*. Esta permite obtener el cuadrado de cualquier número real positivo.

$$
\begin{array}{l}
cuadrado : Rmas \twoheadrightarrow Rmas \\
\hline
\forall r : Rmas \bullet \\
\quad cuadrado(r) = r * r
\end{array}
$$

Se define la función auxiliar *raíz cuadrada*, que permite obtener la raíz cuadrada de cualquier número real positivo.

$$
\begin{array}{l}
raizcuadrada : Rmas \twoheadrightarrow Rmas \\
\hline
\forall r : Rmas \bullet \\
\quad raizcuadrada(r) * raizcuadrada(r) = r
\end{array}
$$

La función *puntos esfera* toma como argumentos el centro y el radio de una esfera que forma parte de la escena y devuelve el conjunto de puntos que componen dicha esfera. Esta tarea la realiza al identificar el conjunto de puntos (x, y, z) que cumplen con la ecuación implícita de la esfera dada por $(x - centro.x)^2 + (y - centro.y)^2 + (z - centro.z)^2 - radio^2 = 0$.

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

$$
\begin{array}{l}
puntos_esfera : Punto \times Rmas \rightarrow\!\!\!\!\!\!+ \; \mathbb{P}\, Punto \\
\hline
\forall\, centro : Punto;\; radio : Rmas \bullet \\
puntos_esfera(centro, radio) = \\
\quad \{ w : Punto \mid \\
\qquad cuadrado(w.x - centro.x) + \\
\qquad cuadrado(w.y - centro.y) + \\
\qquad cuadrado(w.z - centro.z) - \\
\qquad cuadrado(radio) = 0 \;\}
\end{array}
$$

De manera similar, la función *puntos_cilindro* toma como argumentos el centro, el radio, la orientación y la altura del cilindro para devolver el conjunto de puntos que cumplen con la ecuación implícita del cilindro para una altura dada.

$$
\begin{array}{l}
puntos_cilindro : Punto \times Rmas \times Vector \times Rmas \rightarrow\!\!\!\!\!\!+ \; \mathbb{P}\, Punto \\
\hline
\forall\, centro : Punto;\; radio : Rmas;\; dir : Vector;\; h : Rmas \bullet \\
puntos_cilindro(centro, radio, dir, h) = \\
\quad \{ w : Punto \mid \\
\quad cuadrado(w.x - (centro.x + ((w.x - centro.x) * dir.x + \\
\quad (w.y - centro.y) * dir.y + \\
\quad (w.z - centro.z) * dir.z) * dir.x)) + \\
\quad cuadrado(w.y - (centro.y + ((w.x - centro.x) * dir.x + \\
\quad (w.y - centro.y) * dir.y + \\
\quad (w.z - centro.z) * dir.z) * dir.y)) + \\
\quad cuadrado(w.z - (centro.z + ((w.x - centro.x) * dir.x + \\
\quad (w.y - centro.y) * dir.y + \\
\quad (w.z - centro.z) * dir.z) * dir.z)) + \\
\quad -radio = 0 \wedge \\
\quad 0 \leq (w.x - centro.x) * dir.x + \\
\quad (w.y - centro.y) * dir.y + \\
\quad (w.z - centro.z) * dir.z \leq h \}
\end{array}
$$

La función *puntos_cono* toma como argumentos el centro, la proporción de radio por unidad de altura, la orientación y la altura del cono. Dicha función determina y retorna el conjunto de puntos que cumplen con la ecuación implícita del cono para una altura dada.

$$
\begin{array}{l}
puntos_cono : Punto \times Rmas \times Vector \times Rmas \rightarrow\!\!\!\!\!\!+ \; \mathbb{P}\, Punto \\
\hline
\forall\, centro : Punto;\; prop : Rmas;\; dir : Vector;\; h : Rmas \bullet \\
puntos_cono(centro, prop, dir, h) = \\
\quad \{ w : Punto \mid \\
\quad prop * ((w.x - centro.x) * dir.x + \\
\quad (w.y - centro.y) * dir.y + \\
\quad (w.z - centro.z) * dir.z) - h * prop = 0 \wedge \\
\quad 0 \leq (w.x - centro.x) * dir.x + \\
\quad (w.y - centro.y) * dir.y + \\
\quad (w.z - centro.z) * dir.z \leq h \}
\end{array}
$$

La siguiente función retorna el conjunto de puntos que componen un rayo matemático. Este conjunto de puntos se utilizará para determinar la intersección entre el rayo y los elementos de la escena.

$$
\begin{array}{l}
puntos_de_rayo : Rayo \rightarrow\!\!\!\!\!\!+ \; \mathbb{P}\, Punto \\
\hline
\forall\, ray : Rayo \bullet \\
puntos_de_rayo(ray) = \\
\bigcup \{ t : Rmas \bullet \\
\quad \{ temp : Punto \mid \\
\quad temp.x = ray.p.x + (t * ray.v.x) \wedge \\
\quad temp.y = ray.p.y + (t * ray.v.y) \wedge \\
\quad temp.z = ray.p.z + (t * ray.v.z) \} \}
\end{array}
$$

La función *puntos_de_escena* es la encargada de transformar la escena descrita por medio de elementos geométricos y convertirla en una escena descrita por el conjunto de todos los puntos que forman parte de

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

los objetos geométricos que componen la escena.

$$
\begin{array}{l}
puntos_de_escena == (\lambda Escena \bullet \\
\bigcup\{e : elementos;\ t : Tipo;\ ps : \mathbb{P}\ Punto \\
\quad \mid de_que_tipo(e) = t \land \\
\quad ps = \textbf{if}\ (t = soy_esfera)\ \textbf{then} \\
\qquad puntos_esfera(mi_centro(e), mi_radio(e)) \\
\quad \textbf{else if}\ (t = soy_cilindro)\ \textbf{then} \\
\qquad puntos_cilindro(mi_centro(e), \\
\qquad mi_radio(e), mi_vector(e), mi_altura(e)) \\
\quad \textbf{else if}\ (t = soy_cono)\ \textbf{then} \\
\qquad puntos_cono(mi_centro(e), mi_proporcion(e), \\
\qquad mi_vector(e), mi_altura(e)) \\
\quad \textbf{else}\ \varnothing \\
\quad \bullet\ ps\})
\end{array}
$$

La función *intersección* toma como argumentos la escena descrita en puntos y el rayo matemático descrito en puntos para retornar el conjunto de puntos en los cuales el rayo tiene intersección con los elementos de la escena.

$$
\begin{array}{l}
interseccion : \mathbb{P}\ Punto \times \mathbb{P}\ Punto \nrightarrow \mathbb{P}\ Punto \\
\hline
\forall\ ps, qs : \mathbb{P}\ Punto \\
\bullet\ interseccion(ps, qs) = \\
\qquad ps \cap qs
\end{array}
$$

En el algoritmo de *ray tracing* se requiere conocer únicamente la intersección más cercana al origen del rayo. Dado que que la función interseccion devuelve un conjunto de puntos, es necesario determinar de este conjunto de puntos, cuál es el que se encuentra más cerca al origen del rayo. Esto se realiza mediante la función reducción, la cual recibe como argumentos el conjunto de puntos con los cuales el rayo tuvo intersección así como el origen del rayo y devuelve el punto más próximo al origen del rayo.

$$
\begin{array}{l}
reduccion : \mathbb{P}\ Punto \times Punto \nrightarrow Punto \\
\hline
\forall\ ps : \mathbb{P}\ Punto;\ ojo : Punto \bullet \\
reduccion(ps, ojo) = \\
\quad (\mu\ temp : ps;\ t : Rmas \\
\quad \mid t = min\ \{raizcuadrada(cuadrado(temp.x - \\
\quad ojo.x) + cuadrado(temp.y - ojo.y) + \\
\qquad cuadrado(temp.z - ojo.z))\} \bullet temp)
\end{array}
$$

En el contexto del algoritmo de *ray tracing* es necesario conocer además de la intersección más cercana, a cuál elemento corresponde dicha intersección. La función de_ que elemento permite identificar a cuál elemento pertenece un punto específico.

$$
\begin{array}{l}
de_que_elemento == (\lambda Escena;\ p : Punto \bullet \\
\bigcup\{e : elementos;\ t : Tipo;\ ps : \mathbb{P}\ Punto;\ e_int : \mathbb{P}\ Elemento \\
\quad \mid de_que_tipo(e) = t \land \\
\quad ps = \textbf{if}\ (t = soy_esfera)\ \textbf{then} \\
\qquad puntos_esfera(mi_centro(e), mi_radio(e)) \\
\quad \textbf{else if}\ (t = soy_cilindro)\ \textbf{then} \\
\qquad puntos_cilindro(mi_centro(e), mi_radio(e), \\
\qquad mi_vector(e), mi_altura(e)) \\
\quad \textbf{else if}\ (t = soy_cono)\ \textbf{then} \\
\qquad puntos_cono(mi_centro(e), mi_proporcion(e), \\
\qquad mi_vector(e), mi_altura(e)) \\
\quad \textbf{else}\ \varnothing \\
\quad \land\ e_int = \textbf{if}\ (p \in ps)\ \textbf{then} \\
\qquad \{e\} \\
\quad \textbf{else}\ \varnothing \\
\quad \bullet\ e_int\})
\end{array}
$$

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

Finalmente planteamos el esquema *Calcular_ Intersección*, que define el cálculo de la intersección rayo/objeto. Dicho esquema toma como entrada el rayo, el ojo de la escena y retorna el elemento que tuvo la intersección más cercana al origen del rayo, como se requiere en el algoritmo de *ray tracing.*

$$
\begin{array}{l}
\textit{Calcular_Interseccion} \\
\Xi Escena \\
rayo? : Rayo \\
ojo? : Punto \\
int! : Punto \\
e! : \mathbb{P}\ Elemento \\
\hline
int! = reduccion(interseccion(puntos_de_rayo(rayo?), \\
puntos_de_escena(\theta Escena)), ojo?) \\
e! = de_que_elemento(\theta Escena, int!)
\end{array}
$$

IV. CONCLUSIÓN

En este caso de estudio mostramos una especificación abstracta del cálculo de la intersección rayo/objeto requerido por el algoritmo de *ray tracing* mediante la notación Z. El cálculo de la intersección rayo/ objeto corresponde a una de las partes críticas del algoritmo de ray tracing. La investigación previa se ha concentrado en la descripción procedimental de esta parte del algoritmo, mas no fue posible encontrar una especificación precisa y declarativa del cálculo de la intersección rayo/objeto. Al definir este componente del algoritmo mediante una notación formal, como Z, permite tener un mejor conocimiento sobre el funcionamiento y el comportamiento del cálculo de la intersección rayo/ objeto, lo cual abre la posibilidad de atacar el problema de rendimiento que esta parte del algoritmo conlleva de manera más informada, con miras a obtener programas verificables que reduzcan el tiempo de procesamiento requerido para calcular las intersecciones.

V. REFERENCIAS

[1] J.-R. Abrial, S.A. Schuman, and B. Meyer. Specification language. On the Construction of Programs, 1980.

[2] Melvin J. Alvarado, Diego Valderde, Randolph Stein- vorth, and Francisco J. Torres-Rojas. RTUCR: Hardware para Ray Tracing. In Tiempo Compartido, Volumen 8, Número 1, Tiempo Compartido, pages 6-13, Cartago, Cartago, Costa Rica, 2008. Escuela de Ingeniería en Computación, Tecnológico de Costa Rica.

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

[3] Jeff Atwood. Real-time raytracing, March 2008.

[4] Jonathan P. Bowen. Provably Correct Systems: Community, Connections, and Citations, pages 313-328. Springer International Publishing, Cham, 2017.

[5] Jamis Buck. The Ray Tracer Challenge: A Test-Driven Guide to Your First 3D Renderer (Pragmatic Bookshelf).Pragmatic Bookshelf, 2019.

[6] George S. Carson. The specification of computer graphics systems. IEEE Computer Graphics and Applications, 3(6):27-30, 32-36, 38-41, September 1983.

[7] D. A. Duce. Formal specification of graphics software. In Rae A. Earnshaw, editor, Theoretical Foundations of Computer Graphics and CAD, pages 543-574, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[8] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. Computer Graphics: Principles and Practice (3rd Edition). Addison-Wesley Professional, 3 edition, 7 2013.

[9] Tiffany C. Inglis. Ray tracing workshop, 2012.

[10] ISO/IEC. ISO/IEC 13568:2002. Information Technology— Z Formal Specification Notation— Syntax, Type System and Semantics. ISO/ IEC, 2002.

[11] Jonathan Jacky. The Way of Z: Practical Programming with Formal Methods. Cambridge University Press, USA,1996.

[12] Won-Jong Lee, Seok Joong Hwang, Youngsam Shin, Jeong-Joon Yoo, and Soojung Ryu. An efficient hybrid ray tracing and rasterizer architecture for mobile gpu. In SIGGRAPH Asia 2015 Mobile Graphics and Interactive Applications, SA '15, pages 2:1-2:4, New York, NY, USA, 2015. ACM.

[13] Won-Jong Lee, Youngsam Shin, Jaedon Lee, Jin- Woo

Kim, Jae-Ho Nah, Seokyoon Jung, Shihwa Lee, HyunSang Park, and

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

Tack-Don Han. Sgrt: A mobile gpu architecture for real-time ray tracing. In Proceedings of the 5th High-Performance Graphics Conference, HPG '13, pages 109-119, New York, NY, USA, 2013. ACM.

[14] Chris Matthews and Paul A. Swatman. Fuzzy concepts

and formal methods: A fuzzy logic toolkit for z. In ZB 2000: Formal Specification and Development in Z and B, pages 491-510, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[15] Gerard O'Regan. Concise Guide to Formal Methods - Theory, Fundamentals and Industry Applications. Undergraduate Topics in Computer Science. Springer, 2017.

[16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. Physically Based Rendering, Third Edition: From Theory to Implementation. Morgan Kaufmann, 3 edition, 11 2016.

[17] Daniel Plagge and Michael Leuschel. Validating Z specifications using the ProB animator and model checker. In Jim Davies and Jeremy Gibbons, editors, Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings, volume 4591 of Lecture Notes in Computer Science, pages 480-500. Springer, 2007.

[18] Mike Seymour. The art of rendering (updated), April 2012.

[19] J Michael Spivey. The Z notation, 2nd. Ed. Prentice Hall International, 1992.

[20] Kevin Suffern. Ray Tracing from the Ground Up. A K Peters/CRC Press, 9 2007.

[21] Francisco J. Torres-Rojas. Técnicas Básicas de Ray Tracing para Superficies Cuadráticas. In Tiempo Compartido, Volumen 6, Número 4, Tiempo Compartido, pages 34-41, Cartago, Cartago, Costa Rica, 2006. Escuela de Ingeniería en Computación, Tecnológico de Costa Rica.

[22] Marek Vinkler, Vlastimil Havran, and Jiří Bittner. Bounding Volume Hierarchies Versus Kd-trees on Contemporary Many-core

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr

Architectures. In Proceedings of the 30th Spring Conference on Computer Graphics, SCCG ’14, pages 29-36, New York, NY, USA, 2014. ACM.

[23] Ingo Wald. On Fast Construction of SAH-based Bounding Volume Hierarchies. In Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing, RT ’07, pages 33-40, Washington, DC, USA, 2007. IEEE Computer Society.

[24] Yunbo Wang, Chunfeng Liu, and Yangdong Deng. A feasibility study of ray tracing on mobile gpus. In SIGGRAPH Asia 2014 Mobile Graphics and Interactive Applications, SA ’14, pages 3:1-3:5, New York, NY, USA, 2014. ACM.

[25] Alan Watt and Mark Watt. Advanced Animation and Rendering Techniques. ACM, New York, NY, USA, 1991.

[26] Jim Woodcock and Jim Davies. Using Z: Specification, Refinement, and Proof. Prentice- Hall, Inc., USA, 1996.

[27] Henri Ylitie, Tero Karras, and Samuli Laine. Efficient incoherent ray traversal on gpus through compressed wide bvhs. In Proceedings of High Performance Graphics, HPG ’17, pages 4:1-4:13, New York, NY, USA, 2017.

Los artículos de la Revista Technology Inside del Colegio de Profesionales en informática y Computación, Costa Rica, se comparten bajo términos de la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional. Las autorizaciones adicionales a las aquí delimitadas se pueden obtener en el correo: comite.editorial@cpic.or.cr