UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA

EN INFORMÁTICA

PROYECTO FIN DE CARRERA

YAReW: Sistema Asistente al Proceso de Render basado en Técnicas de Data Mining

Lorenzo Manuel López López

Diciembre, 2007

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Informática

PROYECTO FIN DE CARRERA

Autor: Lorenzo Manuel López López
Director: Carlos González Morcillo

Diciembre, 2007.

Este documento fue compuesto con LaTeX. Imágenes generadas con OpenOffice y GIMP.

TRIBUNAL:

Presidente:
Vocal:
Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

Elegir una buena configuración de los parámetros de entrada de cualquier motor de render basado en iluminación global no es una tarea fácil. La calidad de los resultados producidos y el tiempo de cómputo requerido por el proceso son dos medidas que dependen fuertemente de la configuración de valores escogida. Además, cada implementación define sus propios parámetros con unos valores de entrada y un significado concreto. Esto supone que la búsqueda de una buena configuración de render se lleva a cabo normalmente a través de un proceso de prueba y error muy costoso en tiempo para el usuario.

El sistema propuesto con este proyecto es un asistente orientado a proporcionar soporte a la decisión al usuario en términos de configuraciones de render que maximizan la calidad de los resultados producidos tratando de minimizar el tiempo de cómputo requerido.

Abstract

Choosing a good configuration of the input parameters for a any global illumination based render engine is not an easy task. The quality of the synthesised image, as well as the CPU time spent in the process, depends to a great extent on the values of the input variables of the render engine which is to be used. Besides, those variables and their values are completely dependent on each implementation, which means that usually a trial-error process is the only way to learn how to find the best configuration of input parameters for each situation.

The system which has been developed along with this project is a data mining based rendering wizard which aims to give to the user decision support consisting on rendering settings which try to maximise the quality of the synthesised image and minimise the CPU time required by the process.

A papá y a mamá.
Por vuestro amor, entrega e ilusión.

Agradecimientos

Muchas han sido las personas que han contribuido a lo largo de toda mi vida a alcanzar tan importante hito, sin embargo el espacio disponible aquí es limitado, por lo que espero que las no incluidas sepan entenderlo.

A Carlos González, por la inspiración, los ánimos, las ideas, por su gran ayuda en el desarrollo de este proyecto, por la confianza que ha depositado en mi desde el primer momento y sobre todo, por ser un gran amigo a la vez que un gran director. Al grupo de investigación Oreto, por permitirme utilizar sus instalaciones durante la parte más intensiva del desarrollo de este proyecto y rodearme de tan genial ambiente de trabajo. A Javier Albusac, por su ayuda en la revisión de este documento. A José Ángel (Blankito) Mateos, por haber sido mi gran compañero y hermano desde el principio hasta el final de esta aventura. ¡Qué bien nos lo hemos pasado!. A Maribel, por todo el apoyo y el amor. Te has convertido en un pilar muy importante en mi vida. A Papá, porque con tu esfuerzo esto se ha hecho posible. A mis hermanos, por todo lo que nos queremos. Y a Maribelita, por la comprensión y todo el amor que me das, a pesar de todo el tiempo que esto te ha robado.

A Mamá, por todas las fuerzas que me das cada día. Siempre estarás conmigo.

Índice general

Índice de figuras

Índice de cuadros

Índice de algoritmos

Capítulo 1

Introducción

1.1. Motivación

En los últimos años estamos asistiendo a un auténtico auge en la utilización de gráficos por computador. Atrás quedaron aquellos tiempos en que su producción era extremadamente costosa y tan sólo era posible su utilización en el ámbito científico, para propósitos de visualización de información o simulación. Sin embargo, hoy en día son utilizados en una gran variedad de contextos diferentes, como en la televisión, el marketing, el diseño asistido por computador y el ocio, donde tiene sus dos pilares más sólidos en las industrias del cine y los videojuegos. Aún así, el proceso de creación de gráficos por computador de calidad fotorrealista es muy costoso y una gran parte de la comunidad científica sigue enfocada en la optimización de los distintos aspectos que intervienen en su producción.

La **síntesis de imagen realista** tiene como objetivo la generación de imágenes bidimensionales a partir de descripciones abstractas de modelos en tres dimensiones, con una calidad que las haga difícilmente distinguibles de las imágenes reales (fotografías) incluso para ob-

servadores expertos[1].

El proceso de generar una imagen a partir de un modelo abstracto en tres dimensiones está dividido en varias etapas: modelado de geometrías, la definición de materiales y texturas, la colocación de la cámara y fuentes de iluminación virtuales y por último, el renderizado. Los algoritmos de render toman como entrada una descripción de geometría, materiales, texturas, fuentes de luz y cámaras virtuales y producen una imagen o secuencia de imágenes (en el caso de una animación) como salida. Aunque actualmente existen multitud de algoritmos de render diferentes, casi todos ellos tratan de resolver el mismo problema: **la simulación del comportamiento físico de la luz**. De este grupo, existe un conjunto denominado *algoritmos de iluminación global* que se caracterizan por tener en cuenta tanto la iluminación directa (de las fuentes de luz a los objetos de la escena), como la indirecta (entre objetos de la escena).

Un factor común que tiene cualquier algoritmo de iluminación global deriva de la complejidad inherente al proceso de simulación del comportamiento físico de la luz: son procedimientos muy costosos en términos computacionales. Diferentes algoritmos toman diferentes aproximaciones para resolver el problema, lo que supone la existencia de un amplio rango de soluciones que van desde las más sencillas en su implementación y rápidas en su ejecución, hasta las más complejas y precisas en los resultados.

La etapa de renderizado es comúnmente vista como uno de los cuellos de botella más importantes en el proceso de síntesis de imagen realista. La síntesis de una sola imagen puede requerir desde varias horas, hasta días de tiempo de procesador, según la complejidad del modelo 3D y la configuración de entrada del motor de render establecida.

Precisamente, la configuración de las variables de entrada del motor de render utilizado es una de las fases más críticas dentro de la etapa de render. El tiempo de ejecución del algoritmo y la calidad del resultado producido dependen en gran parte de la configuración escogida. Además, aunque ciertos parámetros son comunes a varias implementaciones, cada motor define su propio conjunto de parámetros de entrada y el significado de estos es altamente dependiente de cada implementación particular. Algunos ejemplos de los parámetros más comúnmente utilizados son el número de muestras tomadas por fuente de iluminación o la profundidad en la recursividad para el proceso de trazado de rayos.

[1] *"Si parecen gráficos por computador, entonces no son buenos gráficos por computador"*. Jeremy Birn

Escoger una buena configuración de parámetros de entrada para un motor de render determinado no es tarea fácil. Primero, es necesario conocer el significado de cada parámetro en el caso particular de la implementación utilizada. Después, una buena configuración de estos parámetros es normalmente alcanzada mediante un proceso iterativo en el que el usuario consigue una aproximación a la configuración óptima a través de un proceso de prueba y error. En cada iteración se establece una configuración de prueba y es evaluada mediante un proceso de **profiling**[2]. La calidad de la aproximación conseguida por el usuario dependerá de varios factores, como la experiencia del usuario en procesos de render en general, la experiencia con el motor particular utilizado, el número de iteraciones empleadas en el proceso, el grado de entendimiento del significado de cada parámetro y la complejidad de la escena a procesar.

Experiencia	Tiempo de Configuración (mins)	Calidad Obtenida
Menos de 1 semana	19	4
1 Semana - 1 Mes	32	2
1 Semana - 1 Mes	60	6
1 Mes - 1 Año	17	2
1 Mes - 1 Año	9	6
1 Mes - 1 Año	15	3
1 Mes - 1 Año	20	5
1 Año - 2 Años	25	9
1 Año - 2 Años	33	3
1 Año - 2 Años	3	5
1 Año - 2 Años	53	8
1 Año - 2 Años	90	7
1 Año - 2 Años	60	9

Cuadro 1.1: Tiempos de configuración y resultados obtenidos en la escena "*Simple*"

Para tener una evidencia de cuánto tiempo es empleado por distintos usuarios en configurar un motor de render determinado, se ha realizado un experimento [5] en el cual se ha solicitado a la comunidad de usuarios del motor de render *Yafray* [6] la configuración y render de dos escenas de interior diseñadas para forzar la necesidad de una correcta iluminación indirecta. La figura 1.1 muestra algunos resultados de diferente calidad obtenidos por los individuos que han realizado el experimento. La columna de la izquierda en esta figura muestra los resultados de la escena etiquetada como "simple" y la de la derecha, resultados

[2]El *profiling* consiste en ejecutar el proceso de render a una baja resolución, con el objeto de obtener una estimación del resultado final sin la necesidad de ejecutar el proceso de render completo.

Experiencia	Tiempo de Configuración (mins)	Calidad Obtenida
Menos de 1 semana	18	5
1 Semana - 1 Mes	30	5
1 Semana - 1 Mes	15	4
1 Semana - 1 Mes	45	4
1 Mes - 1 Año	18	7
1 Mes - 1 Año	210	3
1 Año - 2 Años	15	5
1 Año - 2 Años	42	6
1 Año - 2 Años	34	6
1 Año - 2 Años	18	4
1 Año - 2 Años	7	6
1 Año - 2 Años	20	7
1 Año - 2 Años	60	9

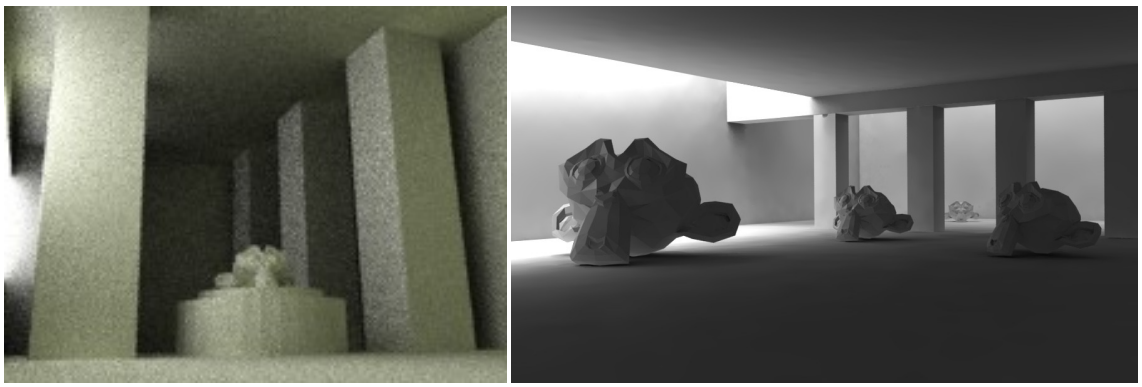
Cuadro 1.2: Tiempos de configuración y resultados obtenidos en la escena "*Compleja*"

de la "compleja".

Ambas escenas son muy similares en cuanto al número de polígonos, fuentes de iluminación, materiales y texturas. Además, en ambas la iluminación proviene de fuentes de luz externas a la escena. La diferencia más notable entre ellas es que una (a partir de ahora la llamaremos escena *simple*) requiere un menor esfuerzo de configuración para conseguir un buen resultado, mientras que la otra (a partir de ahora, escena *compleja*) dificulta en mayor grado la obtención de buenos resultados. Esto se debe a que los caminos que debe seguir la luz para llegar desde la cámara virtual hasta cada fuente son más complejos en una escena que en la otra.

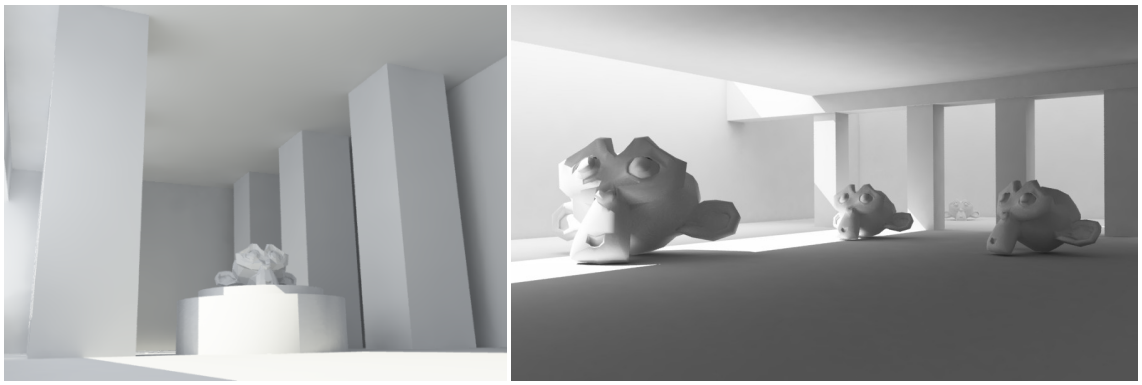
La decisión de que las únicas fuentes de iluminación provengan del exterior de la escena ha sido tomada para forzar la necesidad de un mayor esfuerzo en la configuración. La muestra de usuarios que han participado en este experimento (ver tablas 1.1 y 1.2) poseen diferente experiencia en la utilización de Yafray como motor de render que van desde menos de una semana, hasta dos años.

La figura 1.2 muestra los resultados del experimento realizado. El atributo *Tiempo de Configuración* se refiere al tiempo requerido por los usuarios para establecer los valores de los parámetros de entrada al motor de render. Este tiempo está tomado en minutos. El atributo *Calidad* es una medida que puede tomar valores en el intervalo [1,10], donde el valor 1 significa la *menor* calidad y el 10 la *mayor* calidad. Para etiquetar la calidad de cada resultado



(a) Resultados de baja calidad

(b) Resultados de media calidad



(c) Resultados de alta calidad

Izquierda: Escena “Simple”. **Derecha:** Escena “Compleja”.

Figura 1.1: Algunos resultados de diferente calidad obtenidos en el experimento sobre configuración de render realizado (ver versión a color en C.2).

recibido se ha recurrido al juicio de un experto.

Como puede verse en la figura 1.2, independientemente de la experiencia de cada usuario

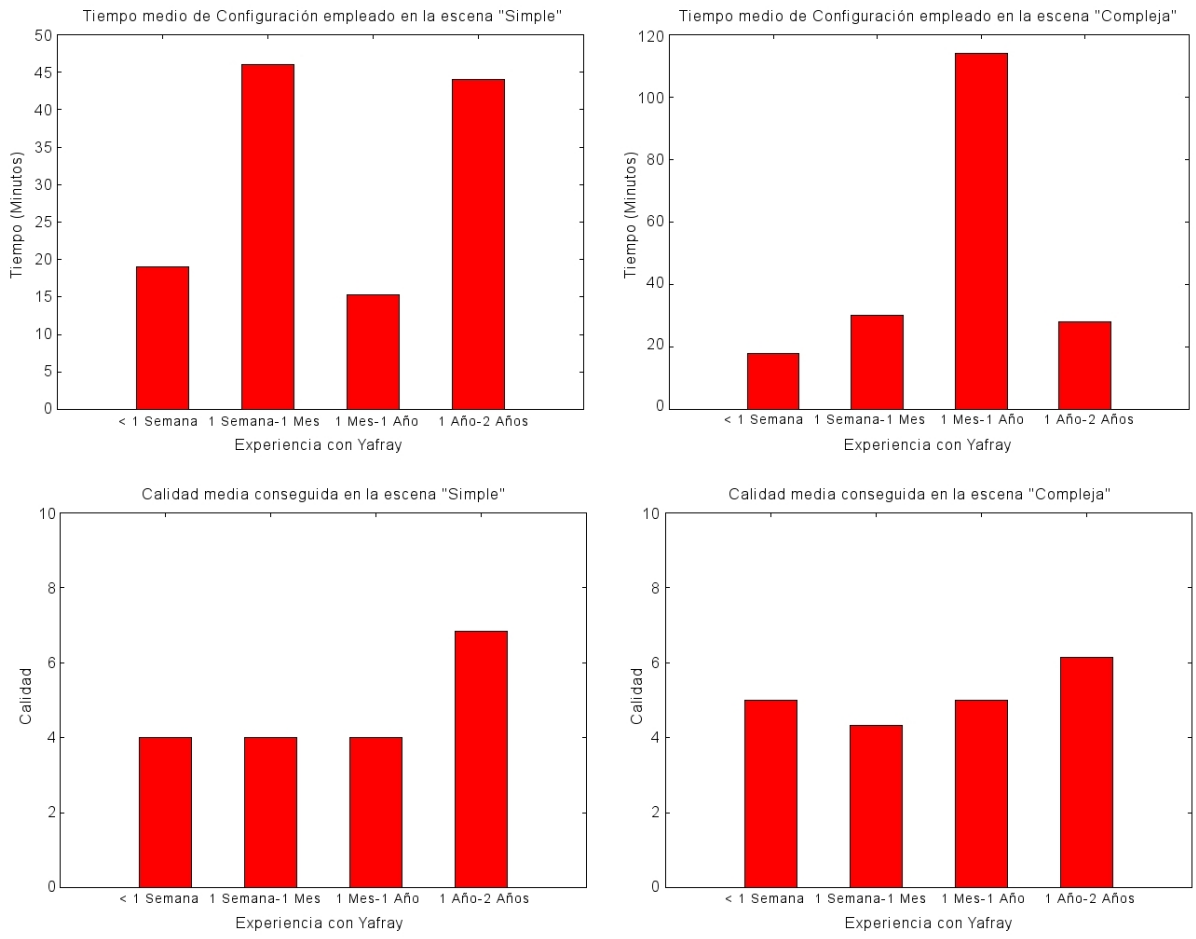


Figura 1.2: Tiempo medio de configuración y calidad media conseguida para las escenas *simple* y *compleja* en el experimento realizado en *Magarro*

en el campo de gráficos en general y en la utilización de Yafray como motor de render en particular, el tiempo medio requerido por un usuario en establecer unos valores adecuados para los parámetros de render ha sido entre treinta y sesenta minutos. Además, la calidad media obtenida en cada caso está en torno a un nivel *aceptable*, lo que pone en evidencia la dificultad de conseguir resultados de alta calidad. Por este motivo, normalmente los usuarios se dan por satisfechos con una calidad *aceptable* a cambio de invertir menos tiempo en encontrar una buena configuración de render.

1.2. Terminología

En esta sección se presenta una colección de términos clave que son ampliamente utilizados a través del presente documento.

- **Render**: Proceso síntesis de imagen a partir de una descripción abstracta de una escena en tres dimensiones. El objetivo principal de este proceso es la determinación del color de cada uno de los píxeles que forman la imagen bidimensional correspondiente a la proyección del plano de una cámara virtual en un modelo tridimensional.

- **Escena o Modelo 3D**: Descripción abstracta de un modelo tridimensional. Esta descripción consiste en la definición de geometrías, materiales, texturas y cámaras y fuentes de iluminación virtuales. El formato de estas descripciones es dependiente del software de modelado o render utilizado.

- **Parámetros de render**: Conjunto de características definidas por cada software de render que especifican ciertos aspectos relativos a su implementación. Estas características afectan en gran medida al tiempo requerido por el proceso de render y a la calidad del resultado obtenido. Aunque existen ciertas características cuyo significado es ampliamente aceptado, cada software particular define un conjunto propio de parámetros y el significado completo de cada uno de ellos es muy dependiente de cada implementación en particular. Algunos ejemplos son *número de muestras por píxel* o *profundidad en la recursividad para rayos de iluminación directa o indirecta*.

- **Decisión**: Se define una decisión como la toma de una elección sobre un conjunto de alternativas.

- **Soporte a la decisión**: Un proceso de soporte a la decisión es aquel que facilita al usuario la toma de una decisión sobre un conjunto de alternativas posibles. Para ello, dicho proceso debe contemplar ciertos aspectos como el entendimiento del problema, la colección y verificación de información relevante, la identificación de alternativas y la anticipación y evaluación de las consecuencias derivadas de la toma de una decisión sobre otra.

- **Modelo**: En el vocabulario propio de la Minería de Datos o el Aprendizaje Máquina, el modelo hace referencia a la función descrita por un conjunto de datos o instancias (ver 3.1.2).
- **Dominio**: En el vocabulario propio de la Minería de Datos o el Aprendizaje Máquina, el dominio hace referencia al conjunto de atributos que describen cada una de las instancias del modelo, incluido el atributo objetivo o atributo de clase.
- **Dominio de Tiempo**: Dominio definido en el que el atributo objetivo es el tiempo de render. Ya que el tiempo es un atributo continuo, los algoritmos utilizados para actuar sobre este dominio deben ser apropiados para problemas de *regresión*.
- **Dominio de la Calidad**: Dominio definido en el que el atributo objetivo es la calidad de imagen producida por una instancia de render. La calidad es un atributo categórico, por lo que los algoritmos utilizados para actuar sobre este dominio deben ser apropiados para resolver problemas de *clasificación*.

1.3. Estructura del documento

Los objetivos perseguidos en este proyecto y las hipótesis de trabajo sobre las que se ha guiado su desarrollo son descritos con un mayor grado de detalle en el capítulo 2. En el capítulo 3 se describe el estado de la cuestión en temas de Minería de Datos y Aprendizaje Automático, y Síntesis de Imagen Realista. A continuación se detalla el proceso de desarrollo seguido para construir el sistema propuesto en el capítulo 4. Los resultados de las pruebas desarrolladas sobre el sistema desarrollado son detallados en el capítulo 5, seguidos de un resumen de las conclusiones obtenidas tras la ejecución de este proyecto y algunas ideas acerca de posibles líneas de investigación futuras en el capítulo 6.

Para finalizar, se incluyen tres apéndices. En el apéndice A se muestra el código fuente de algunos de los componentes del sistema desarrollado. En el apéndice B se muestran las imágenes resultado de renderizar las escenas incluidas en la base de conocimiento de la versión del sistema presentada con este proyecto. Por último, en el apéndice C se muestra la versión a color de algunas de las figuras incluidas en este documento.

Capítulo 2

Objetivos e Hipótesis de Trabajo

Tal como se ha visto en el capítulo 1, según el experimento realizado, el tiempo medio requerido por un usuario en establecer una buena configuración de render ha estado entre los treinta y los sesenta minutos. Además, la calidad media obtenida ha estado en torno a un nivel *aceptable* (ver figura 1.2). Por lo tanto, el objetivo del presente proyecto es el desarrollo de un sistema asistente al proceso de render de gráficos 3D capaz de proporcionar al usuario soporte a la decisión en los siguientes términos:

- **Proporcionar sugerencias de configuraciones de parámetros de render que maximicen la calidad de los resultados obtenidos y minimicen el tiempo de ejecución de los algoritmos de render utilizados**: El sistema debe tener la capacidad de sugerir configuraciones de render que maximicen la relación $\frac{\text{Calidad de Resultado}}{\text{Tiempo de Render}}$. Estas sugerencias deben realizarse en base a un grado de libertad especificado por el usuario, mediante el cual el sistema ofrecerá configuraciones que se parezcan en mayor o menor medida a la configuración proporcionada por el usuario. Esta medida de libertad se denominará *grado de afinidad con el usuario* y tomará valores en el intervalo [0.0,1.0]. Un valor de afinidad 1.0 indicará al sistema que debe proporcionar aquella configuración encontrada que se parezca en mayor grado a la proporcionada por el usuario y a la vez maximice la relación definida. Por el contrario, un valor 0.0 indicará al sistema que tiene plena libertad para sugerir aquella configuración que maximice dicha relación ignorando por completo la configuración de render proporcionada por el usuario.

- **Estimación del tiempo de render y la calidad del resultado prevista para una escena y una configuración de parámetros de render proporcionados por el usuario**: El sistema debe ser capaz de ofrecer estimaciones del tiempo de render y la calidad del resultado obtenido a partir de una escena y una configuración de parámetros de render determinados por el usuario. Esta información es de gran utilidad, ya que permite obtener una estimación del coste y el rendimiento del proceso de render sin la necesidad de utilizar técnicas de *profiling*, las cuales tienen un coste asociado que, aunque substancialmente menor que el del proceso de render en sí, no es despreciable.

La hipótesis de trabajo mediante la cual se pretenden conseguir los objetivos expuestos es que es posible analizar los datos obtenidos de la realización de un gran número de instancias de render para extraer ciertas relaciones no triviales que den lugar a patrones de información útiles para proporcionar soporte a la decisión al usuario. Así, se hace necesaria la generación de un gran número de ejemplares del proceso de render y la utilización de técnicas y procedimientos desarrollados en los campos de Minería de Datos y Aprendizaje Máquina.

A partir de los objetivos principales, se definen una serie de requisitos que deben cumplirse:

- **Construcción de una plataforma de análisis de datos y soporte a la decisión genérica**: Debido al gran número de paquetes de software de modelado y render de gráficos 3D disponibles en el mercado, es importante conseguir una independencia entre el núcleo del sistema y el software utilizado por el usuario. Para ello, se definen los siguientes aspectos en los que el sistema debe ser genérico:

  - *Genérico respecto a los métodos de render empleados*: El sistema debe tener la capacidad de cumplir los objetivos citados sin importar el método de render empleado. Para conseguir este aspecto, el sistema debe ofrecer sugerencias y estimaciones basándose tan sólo en la experiencia de la realización de un conjunto de instancias del proceso de render sobre distintas escenas y con diferentes configuraciones de parámetros de entrada.
  - *Genérico respecto a los métodos de aprendizaje empleados*: El sistema debe permitir la incorporación de nuevos métodos de aprendizaje. Esto se consigue me-

diante la representación en un formato estándar de la base de conocimiento, de tal forma que cualquier método pueda hacer uso de esta para proporcionar estimaciones sobre nuevos datos de entrada.

- *Genérico respecto a las variables de render empleadas*: Debido a que cada software de render define sus propios parámetros de entrada, es importante conseguir una independencia entre el conjunto de variables que definen las instancias de render y el núcleo del sistema en sí.

Estas propiedades hacen que el sistema a desarrollar sea **adaptable** a nuevos métodos de render, software de usuario utilizado y métodos de aprendizaje o minería de datos empleados.

- **Instanciación de la plataforma creada para su utilización con el software empleado en el desarrollo de este proyecto**: En el desarrollo de este proyecto se utilizarán un software de desarrollo de gráficos 3D y un motor de render específicos. La plataforma genérica descrita en el punto anterior debe ser adaptada para su utilización con este software en particular. En concreto, el software de desarrollo de gráficos será Blender [4] y el motor de render empleado será Yafray [6].

- **Desarrollar una plataforma robusta**: El grado de precisión en las sugerencias ofrecidas por el sistema estará condicionado en un alto grado por la expresividad de la base de conocimiento manejada. Se pretende la construcción de una base de conocimiento con una alta diversidad de modelos 3D que permita al sistema ofrecer estimaciones con un alto grado de precisión para un espectro de escenas proporcionadas por el usuario relativamente amplio. No obstante, el sistema debe tener la capacidad de ofrecer sugerencias en todo momento, aunque estas no sean muy precisas si los modelos proporcionados por el usuario son de muy distinta naturaleza a los contenidos en la base de conocimiento.

- **Desarrollo de un sistema multi-plataforma**: El sistema desarrollado debe permitir su ejecución en una gran variedad de plataformas software y hardware diferentes. Para conseguir este aspecto, se utilizarán tecnologías multi-plataforma para su desarrollo.

- **Desarrollo del sistema conforme a estándares abiertos**: Para conseguir adaptabilidad del sistema a nuevos métodos de aprendizaje y análisis de datos, se utilizarán estándares abiertos para la representación del conocimiento manejado por el sistema.
- **Desarrollo del sistema utilizando software de código abierto**: Todo el desarrollo del presente proyecto se llevará a cabo mediante la utilización de tecnologías de código abierto. Con este aspecto se consigue una continuidad del software desarrollado y su posible integración con alguno de las aplicaciones utilizadas.

Capítulo 3

Antecedentes, Estado de la Cuestión

3.1. Minería de Datos y Aprendizaje Máquina

3.1.1. Introducción

Los avances ocurridos en los últimos años en las tecnologías de almacenamiento han hecho posible la acumulación de grandes cantidades de datos a un coste muy razonable. Esto, unido a los avances en las comunicaciones y dispositivos de recogida de información ubicuos,

ha posibilitado que cualquier individuo u organización pueda adquirir y acumular grandes volúmenes de datos que pueden ser de utilidad para generar información útil. Un ejemplo de esto puede verse examinando el funcionamiento de cualquier supermercado. Los puntos de venta almacenan toda la información acerca de los productos que han sido comprados, cuándo y en algunas situaciones incluso por quién han sido comprados. Estos datos no son de mucha utilidad por sí mismos, pero si se pueden procesar para encontrar patrones que describan ciertas situaciones, pueden ser transformados en información muy útil para su propietario.

El proceso de minería de datos se define según [32] como *el proceso automático o semi-automático de descubrimiento de patrones en una colección de datos*. Los patrones buscados son aquellas relaciones no triviales entre los datos que describen situaciones que pueden ser utilizadas para obtener alguna ventaja. En el ejemplo del supermercado, si todos los viernes se registran el mismo número de bolsas de patatas fritas y de latas de cerveza, el propietario de esta información puede sacar provecho de esta situación para aumentar sus beneficios. Para que los patrones obtenidos sean significativos, una condición básica es que los datos deben estar presentes en grandes cantidades.

El tipo de patrones que son buscados por el proceso de minería de datos son aquellos capaces de describir la estructura de una decisión. Estos patrones reciben el nombre de **estructurales** y son importantes porque hacen posible la utilización de esa información para realizar predicciones sobre nuevos datos.

Un aspecto importante en la búsqueda de patrones en los datos es que dicha búsqueda debe ser automática. Esto es algo que se ha venido estudiando a lo largo de los últimos años en el campo del **aprendizaje máquina** [29]. Técnicas como la inducción de árboles de decisión o la clasificación basada en instancias tratan de descubrir un conjunto de patrones o hipótesis relativas a los datos almacenados y que pueden ser generalizadas para predecir nuevos datos. Por tanto, la minería de datos hace uso de técnicas desarrolladas en el campo del aprendizaje máquina para descubrir y describir patrones estructurales descritos por un conjunto de datos que pueden ser utilizados para realizar predicciones sobre nuevos datos en base a ellos.

A continuación se definirá la terminología utilizada a lo largo de este capítulo en la sección 3.1.2. Después se expondrán en detalle algunas de las técnicas de aprendizaje máquina más utilizadas en el proceso de minería de datos como árboles de decisión en la sección 3.1.3,

modelos de aprendizaje basados en instancias en 3.1.4 y modelos de aprendizaje probabilístico en la sección 3.1.5. En la sección 3.1.6 se describirá el modelo de algoritmos genéticos, un método muy utilizado tanto en problemas de aprendizaje, como de búsqueda. Por último se introducirán otras técnicas en la sección 3.1.7 y se definirán algunos métodos para evaluar la eficiencia de estas técnicas en un modelo de datos dado en la sección 3.1.8.

3.1.2. Definiciones

Cualquier método de aprendizaje automático requiere una colección de datos de entrada en forma de conjunto de *ejemplares* cada uno de ellos descrito a través de un conjunto de *atributos*. Además, cada método tratará de *aprender* un *concepto*, también llamado *función objetivo* o *atributo objetivo* [32].

Comúnmente, en aplicaciones de minería de datos se pueden observar básicamente cuatro tipos distintos de aprendizaje. En problemas de *clasificación*, el esquema de aprendizaje es presentado como un conjunto de ejemplares clasificados a partir del cual el objetivo es *aprender* un modo de clasificar nuevos ejemplares o instancias no vistos por el algoritmo. En problemas de predicción numérica o *regresión*, el algoritmo trata de obtener una predicción numérica para un ejemplar no visto a partir de la hipótesis inducida mediante los ejemplares vistos. En *aprendizaje de reglas de asociación*, el algoritmo aprende, además de la clasificación de nuevos ejemplares, cualquier asociación o patrón observado en los atributos que describen los datos. Por último, en problemas de *clustering* el objetivo consiste en aprender una forma de discriminar los datos de entrada según grupos. Sea cual sea el tipo de aprendizaje que implica un problema determinado, el objetivo a aprender se denomina **concepto** y la salida producida por un algoritmo de aprendizaje determinado **descripción del concepto**.

Cualquier método de aprendizaje máquina necesita unos datos de entrada, los cuáles son presentados normalmente en forma de conjunto de **ejemplares** o **instancias**. Una instancia es todo aquel ejemplo individual e independiente del concepto a aprender.

Cada ejemplar de datos se describe mediante sus valores en un conjunto predefinido y fijo de **atributos**. El valor de un atributo para una instancia particular es una medida de la cantidad a la que el atributo hace referencia. Cada atributo puede tomar valores numéricos o discretos,

también llamados nominales. Un **atributo continuo** es todo aquel atributo cuyos valores posibles son numéricos. Un **atributo discreto** es todo aquel que puede tomar valores de un conjunto de tamaño fijo y finito. Los atributos discretos son también llamados **nominales** o **categóricos**.

Cualquier método de aprendizaje toma sus datos de entrada en forma de **conjunto de ejemplares** y representados mediante una matriz en dos dimensiones donde cada fila es un ejemplar de entrada al algoritmo y cada columna corresponde a cada uno de los atributos definidos en el dominio del problema. El último atributo suele ser el correspondiente al *concepto* o *atributo objetivo*. Los problemas que consisten en predecir el valor de un atributo objetivo discreto se denominan **problemas de clasificación**, mientras que aquellos que tratan de estimar un valor numérico reciben el nombre de **problemas de regresión**.

En la tabla 3.1 puede verse un conjunto de datos muy popular entre los repositorios de aprendizaje máquina y minería de datos. Estos datos describen ejemplares de diferentes clases de flores *iris*. Las diferentes clases son *iris setosa*, *iris versicolor* e *iris virginica*. El problema a resolver por tanto consiste en encontrar un modo de clasificar un ejemplar de iris de acuerdo a sus valores de longitud y anchura de pétalo y sépalo.

Longitud de sépalo (cm)	**Anchura de sépalo (cm)**	**Longitud de pétalo (cm)**	**Anchura de pétalo (cm)**	**Tipo**
5.1	3.5	1.4	0.2	*Iris setosa*
5.5	2.3	4.0	1.3	*Iris versicolor*
5.0	3.6	1.4	0.2	*Iris setosa*
7.0	3.2	4.7	1.4	*Iris versicolor*
6.3	2.9	5.6	1.8	*Iris virginica*
4.7	3.2	1.3	0.2	*Iris setosa*
7.1	3.0	5.9	2.1	*Iris virginica*
6.9	3.1	4.9	1.5	*Iris versicolor*
5.8	2.7	5.1	1.9	*Iris virginica*
6.5	2.8	4.6	1.5	*Iris versicolor*
6.3	3.3	6.0	2.5	*Iris virginica*
4.6	3.1	1.5	0.2	*Iris setosa*
4.9	3.0	1.4	0.2	*Iris setosa*
6.4	3.2	4.5	1.5	*Iris versicolor*
6.5	3.0	5.8	2.2	*Iris virginica*

Cuadro 3.1: Conjunto de datos para un problema de clasificación

3.1.3. Árboles de decisión

La inducción de árboles de decisión es uno de los métodos más sencillos y utilizados para construir algoritmos de aprendizaje inductivo. Los árboles de decisión permiten aproximar funciones objetivo con valores discretos o continuos, son muy robustos ante posible ruido en el conjunto de ejemplares de entrada y funcionan bastante bien ante situaciones no deseadas, como valores de atributos desconocidos en ejemplares del conjunto de datos de entrada. Además, una gran ventaja importante de este método es que su representación resulta muy natural para las personas.

Un árbol de decisión [27] toma como entrada un objeto o situación descrita a través de un conjunto de atributos y devuelve una *decisión*, o dicho de otro modo, el valor estimado de la función objetivo para el ejemplar de entrada dado. Los atributos del conjunto de ejemplares de entrada pueden ser discretos o continuos, así como el valor de la función objetivo. Al proceso de aproximar una función objetivo cuyo valor es discreto se le denomina **clasificación**, mientras que si el valor es de naturaleza continua se habla de un problema de **regresión**. De aquí surge la primera taxonomía mediante la cual se clasifican los árboles de decisión según el tipo de problema en el que son aplicados como **árboles de clasificación** y **árboles de regresión** respectivamente. El algoritmo de inducción y el proceso de predicción del valor del atributo objetivo es idéntico en ambos tipos de árboles, las únicas diferencias son la medida utilizada para dividir los nodos intermedios y el tipo de valor devuelto por el algoritmo cuando se trata de estimar un nuevo ejemplar.

La utilización de un árbol de decisión conlleva dos etapas: el entrenamiento del algoritmo y su utilización para estimar nuevos ejemplares. En la etapa de **entrenamiento** o **inducción**, el algoritmo trata de inducir una *hipótesis* que describa bien el dominio representado por los datos de entrenamiento. Esta hipótesis es representada en forma de árbol de decisión y puede ser utilizada en la fase de **estimación** para predecir el valor del atributo objetivo de nuevos ejemplares no vistos durante la etapa de entrenamiento.

Es importante que la hipótesis inducida durante el entrenamiento de un árbol de decisión se ajuste bien a los datos de entrenamiento, aunque esto conlleva algunos problemas si existe ruido en ellos. La hipótesis inducida debe ser consistente con los datos de entrenamiento

porque se supone que estos son una muestra aleatoria del conjunto de todos los posibles datos del dominio. Una hipótesis inducida a partir de un conjunto de datos de entrenamiento de calidad generalizará bien cuando se trate de estimar nuevos ejemplares no vistos en el conjunto de entrenamiento. Sin embargo, si existe ruido en los datos de entrenamiento, el árbol inducirá una hipótesis errónea que no dará buenos resultados a la hora de generalizar para cualquier ejemplar posible del dominio. Este problema recibe el nombre de **sobreajuste** (del inglés *overfitting*).

Cuando un árbol de decisión es inducido correctamente proporciona una serie de reglas que relacionan los valores de los atributos que describen cada ejemplar con los posibles valores que puede tomar la función objetivo. Estas reglas describen la estructura de los patrones encontrados en el conjunto de datos de entrenamiento. A la hora de estimar nuevos ejemplares, estos son sometidos a los test descritos por dichas reglas para llegar a una estimación del valor del atributo objetivo definido.

Para ilustrar el concepto, en la figura 3.1 se muestra el árbol de clasificación inducido a partir del conjunto de datos definido en la tabla 3.1. En este se puede observar cómo se van discriminando los distintos ejemplares desde el nodo raíz (el cuál contiene la misma proporción de ejemplares de las tres clases) hasta los nodos hojas.

Inducción de árboles de decisión

Aunque existe una gran variedad de implementaciones de algoritmos de inducción de árboles de decisión, todas ellas se basan en el algoritmo ID3 [25] desarrollado por J. Quinlan en el cual se utiliza una búsqueda voraz con una aproximación *top-down*. Otros algoritmos muy populares son el C4.5 [26] y su sucesor C5.0, ambos creados también por Quinlan y basados en el núcleo del algoritmo ID3 aunque con algunas extensiones como el manejo de atributos continuos (el algoritmo ID3 original no lo permite) y algunas mejoras en la eficiencia. Ya que el algoritmo ID3 es el más básico, a partir del cual se han desarrollado prácticamente todos los demás, esta explicación se centrará en él.

El algoritmo ID3 induce un árbol de decisión a partir de un conjunto de ejemplares (normalmente llamado **conjunto de entrenamiento**), cada uno de ellos descrito mediante un vector de pares *atributo-valor*.

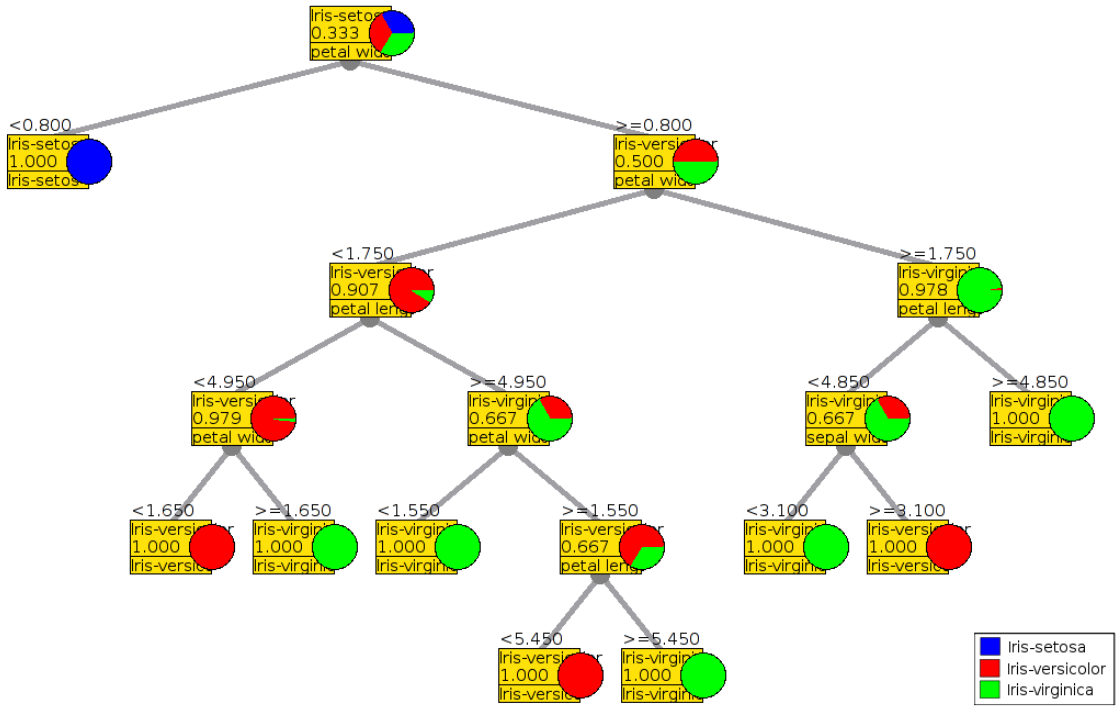


Figura 3.1: Árbol para el problema de clasificación de iris

Comenzando con el nodo raíz, se escoge un atributo y se realiza una partición del conjunto de ejemplares de entrenamiento según sus valores para dicho atributo. Para realizar esta decisión, cada atributo es evaluado mediante un test estadístico cuyo objetivo es determinar cómo de bien es capaz de clasificar los ejemplares de entrenamiento por sí mismo. El atributo que mejor lo haga es seleccionado y se crean tantos nodos hijos como posibles valores tenga dicho atributo. Una vez dividido el nodo en cuestión, se ordenan los ejemplares de entrenamiento de acuerdo con su valor para el atributo seleccionado y el proceso continúa utilizando los ejemplares de cada rama para escoger el mejor atributo para el nuevo nodo. Los atributos ya escogidos no son contemplados más. Se procede de esta forma hasta que todos los atributos han sido utilizados o hasta que todos los ejemplares se han conseguido clasificar de forma correcta, es decir, todas las hojas proporcionan una clasificación para los ejemplares que las alcancen.

Cuando en un nodo determinado no quedan más atributos, pero sí instancias pertenecientes a distintas clases se dice que existe **ruido** en los datos, ya que ejemplares con una

descripción idéntica pertenecen a clases diferentes. Esto puede suceder cuando los atributos no definen de forma completa el concepto que se esta intentando aprender, cuando no hay ejemplares suficientes en el conjunto de datos de entrenamiento para definir con precisión el concepto a aprender o cuando existen ejemplares muy extraordinarios. Una forma sencilla de resolver este problema es mediante el "*voto de la mayoría*". Así, en este tipo de hojas se devuelve aquel valor del atributo objetivo que más veces se repita entre los ejemplares del conjunto de entrenamiento que alcanzan dicha hoja.

Algoritmo 1 ArbolDecision($ejemplares$, $atributos$, $valor\_por\_defecto$)

if $ejemplares$ es vacío **then**
  return $valor\_por\_defecto$
else if todos los elementos de $ejemplares$ pertenecen a la misma clase **then**
  return $clase$
else if $atributos$ es vacío **then**
  return la clase más frecuente en $ejemplares$
else
  $mejor\_atributo \leftarrow$ ElegirMejorAtributo($atributos$, $ejemplares$)
  $arbol \leftarrow$ nuevo árbol con nodo raíz $mejor$
  $m \leftarrow$ clase más frecuente en $ejemplares$
  for all $valor_i$ de $mejor\_atributo$ **do**
    $ejemplares_i \leftarrow$ todo $ejemplar$ cuyo valor para $mejor\_atributo = valor_i$
    $subarbol \leftarrow$ ArbolDecision($ejemplares_i$, $atributos - mejor\_atributo$, m)
    añadir rama a $arbol$ con etiqueta v_i y subárbol $subarbol$
  end for
  return $arbol$
end if

Debido a que los atributos más significativos son utilizados en los primeros nodos, el algoritmo ID3 tiene preferencia por los árboles más pequeños y sencillos que pueden describir el conjunto de entrenamiento. Esto es consistente con el criterio de la **navaja de Occam**[1], el cual dicta que *es preferible la hipótesis más sencilla que sea consistente con los datos*. Esto es debido a que normalmente las hipótesis más sencillas suelen generalizar mejor sobre datos de entrada no incluidos en el conjunto de entrenamiento.

La primera cuestión a abordar en el diseño de este algoritmo es la definición del test estadístico al que cada atributo debe ser sometido para determinar cómo de bien es capaz

[1] Aparentemente, el filósofo inglés William de Occam fue uno de los primeros en discutir esta cuestión alrededor del año 1320 mientras se afeitaba.

de clasificar el conjunto de ejemplares de entrenamiento.La medida utilizada en árboles de clasificación y árboles de regresión difiere. A continuación se definirá la medida de *ganancia de información*, una de las más utilizadas para la evaluación de atributos en árboles de clasificación. Después se definirá su homólogo para los árboles de regresión.

Entropía y ganancia de información para árboles de clasificación

La medida utilizada en árboles de clasificación para calcular cómo de significativo es un atributo a la hora de clasificar los ejemplares de entrenamiento recibe el nombre de *ganancia de información* [23] y su cálculo se basa en la *entropía*, una medida muy comúnmente utilizada en teoría de la información y que caracteriza la impureza de un conjunto arbitrario de ejemplares. Sea S un conjunto de ejemplares que se clasifican según algún concepto objetivo en c clases distintas, la entropía de S en su forma general está especificada por la siguiente expresión:

$$Entropia(S) = \sum_{i=1}^{c} -p_i \cdot log_2 p_i \tag{3.1}$$

Siendo p_i la probabilidad que tiene un ejemplar perteneciente a la clase i de aparecer en el conjunto de todos los ejemplares. En el caso general, si existen c clases diferentes, el valor de la entropía puede ser como máximo $log_2 c$.

Una vez definida la entropía como una medida de la impureza en una colección de datos, se define la *ganancia de información* como la reducción esperada de la entropía causada a partir de la partición de los ejemplares de entrenamiento de acuerdo a un atributo determinado.

$$Ganancia(S, A) = Entropia(S) - \sum_{v \epsilon Valores(A)} \frac{|S_v|}{|S|} \cdot Entropia(S_v) \tag{3.2}$$

La ecuación (3.2) muestra la ganancia de información proporcionada al clasificar todos los ejemplares de un conjunto S mediante el atributo A, donde v es cada uno de los posibles valores que puede tomar el atributo A y la fracción $\frac{|S_v|}{|S|}$ es la proporción de ejemplares de S que pertenecen a S_v, es decir, el subconjunto de S en el que todos los ejemplares contenidos tienen el valor v en el atributo A. La entropía de ambos conjuntos (S y S_v) son calculadas

según la ecuación (3.1). En la figura 3.2 puede verse la función de la entropía en el caso particular de un problema de clasificación binaria.

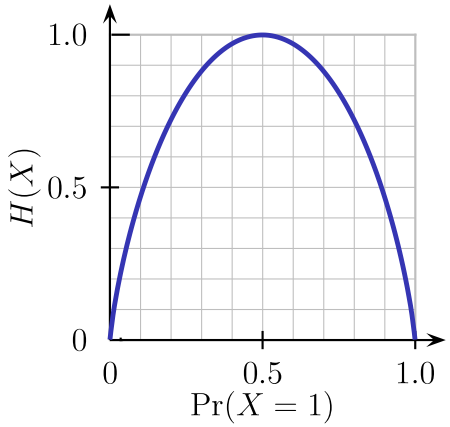


Figura 3.2: Entropía en el caso de una clasificación binaria

Reducción de la Desviación Típica para árboles de regresión

El objetivo de los árboles de regresión es proporcionar una estimación del valor numérico del atributo objetivo para un ejemplar de entrada dado. Ya que el valor a predecir es continuo, no es posible utilizar la ganancia de información para evaluar la calidad de los atributos a la hora de dividir un nodo del árbol. Por este motivo es necesario utilizar una medida que se acomode mejor a la naturaleza continua de la función objetivo en este tipo de problemas, aunque el objetivo de esta medida es el mismo que el de la ganancia de información.

En árboles de regresión se trata de minimizar la desviación típica entre aquellos ejemplares que queden divididos por un nodo determinado. La medida utilizada toma el nombre de *SDR* (*Standard Deviation Reduction*) y es calculada según la ecuación (3.3) [32].

$$SDR = sd(S) - \sum_i \frac{|S_i|}{|S|} sd(S_i) \tag{3.3}$$

donde i es el número de conjuntos de ejemplares que resulta de seleccionar el atributo que está siendo evaluado y S_i es cada uno de esos conjuntos.

El proceso de inducción llegará a un nodo hoja cuando los valores de la función objetivo de todos los ejemplares contenidos en el nodo que está siendo evaluado difieren en una fracción

muy pequeña de la desviación estándar del conjunto de ejemplares inicial.

Predicción en árboles de decisión

Una vez finalizada la fase de entrenamiento, el resultado es un árbol de decisión que describe una hipótesis sobre la descripción del concepto a *aprender*. En este momento se puede utilizar el árbol resultado para predecir el valor del atributo objetivo de nuevos ejemplares.

Cada nodo interno del árbol corresponde con un test que involucra uno de los atributos del dominio. Estos test son reglas que indican qué rama del árbol se debe tomar a continuación según el valor que toma el nuevo ejemplar para el atributo que se está evaluando. Así, comenzando por el nodo raíz, el nuevo ejemplar es sometido a los test especificados por cada uno de los nodos internos que se atraviesen. En un momento dado, el algoritmo llegará a un nodo hoja que especifica el valor a devolver por el algoritmo. Este valor será la clase del ejemplar en el caso de árboles de clasificación, o el valor numérico estimado en el caso de árboles de regresión.

Ruido y Sobreajuste

Si dos o más ejemplares del conjunto de entrenamiento tienen la misma descripción (en términos de los atributos), pero diferentes clasificaciones (o una diferencia muy amplia de valores para el atributo objetivo en problemas de regresión) el árbol de decisión fallará en la estimación de nuevos ejemplares similares porque no sabrá qué valor devolver. Esto es un típico caso de **ruido** en el conjunto de datos de entrenamiento y suele suceder porque no se han incluido todos los atributos necesarios para describir con precisión el concepto a aprender, porque han sido utilizados atributos poco significativos para discriminar un conjunto de ejemplares o simplemente porque algunos ejemplares se han recogido de forma errónea. La solución en estos casos es devolver aquella clasificación que se repita más entre los ejemplares del conjunto de entrenamiento que alcancen dicha hoja o la media aritmética de los valores de los atributos objetivo de dichos ejemplares. Aún así, es muy probable que esta estimación sea incorrecta.

Por otro lado, cuando el conjunto de hipótesis posibles ofrecidas por los datos es muy grande hay que tener cuidado de elegir aquella hipótesis más simple que sea consistente con

los datos. La razón de esto es porque las hipótesis simples tienen una mayor capacidad de generalización ante nuevos ejemplares no vistos durante el entrenamiento. Las hipótesis complejas, aunque sean totalmente consistentes con los datos, tienen un menor grado de adaptación. Cuando una hipótesis consistente con los datos de entrenamiento generaliza mal con ejemplares no vistos se dice que ha ocurrido un problema de **sobreajuste**. Esto significa que la hipótesis inducida es muy dependiente del conjunto de datos de entrenamiento y, aunque sea capaz de describir bien este conjunto, no describe de forma precisa el concepto a aprender.

Una solución muy utilizada en problemas de ruido y sobreajuste es la poda del árbol de decisión. En este aspecto hay dos alternativas. La primera consiste en realizar la poda mientras se está induciendo el árbol. Es decir, en cada nuevo nodo se realiza un test estadístico y según su resultado se seguirá construyendo el árbol por ese camino o se considerará dicho nodo como hoja. Esta aproximación recibe el nombre de **pre-poda** (*pre-pruning*) y tiene la ventaja de ahorrar recursos computacionales ya que no se llega a construir el árbol completo. El otro método consiste en inducir el árbol completo de la forma habitual para después examinar cada nodo intermedio para decidir si se destruye el subárbol que representa o no. Esta aproximación se denomina **post-poda** (*post-pruning*) y, aunque en principio requiere más recursos computacionales que la pre-poda, ofrece unos mejores resultados finales debido a la dificultad de escoger un buen criterio para parar el crecimiento de un árbol en un punto determinado [23].

3.1.4. Aprendizaje basado en instancias

La idea clave sobre la que se basan este tipo de métodos es que *es probable que las propiedades de un punto de entrada particular x sean similares a las de los puntos cercanos a x* [27]. El aprendizaje en este tipo de métodos consiste tan sólo en almacenar los ejemplares del conjunto de entrenamiento. Cuando se presenta un nuevo ejemplar a aproximar, un conjunto de los ejemplares de entrenamiento más similares a él es seleccionado y utilizado para aproximar el valor del atributo objetivo del ejemplar de entrada. Esto significa que casi todo el esfuerzo computacional requerido por el algoritmo se realiza a la hora de aproximar nuevos ejemplares, lo que puede ser una desventaja si el conjunto de ejemplares almacenados es muy

grande. Por este motivo, este tipo de métodos recibe el nombre de *perezosos*.

Una diferencia clave entre los métodos de aprendizaje basado en instancias y otros tipos de métodos como los árboles de decisión es que en lugar de construir una aproximación de la función objetivo completa diseñada para funcionar bien a través de todos los posibles ejemplares, se obtiene una aproximación de la función objetivo **local** a partir de los ejemplares del conjunto de entrenamiento más similares al de entrada. Esto es una gran ventaja cuando la función que se quiere aproximar es muy compleja, pero puede ser descrita a través de una colección de aproximaciones más sencillas.

Una desventaja de este tipo de métodos es que consideran **todos** los atributos de un nuevo ejemplar a aproximar para seleccionar aquellos ejemplares de entrenamiento que son más similares a él. Esto puede no ser una buena idea cuando el valor del atributo objetivo depende realmente de un subconjunto de atributos. Esto difiere de otros métodos, como los árboles de decisión en los que se hace un test de relevancia a todos los atributos con el objetivo de utilizar primero aquellos más importantes. Otra desventaja surge del hecho de que todo el esfuerzo computacional es realizado a la hora de aproximar nuevos ejemplares. A diferencia de otros métodos que construyen una hipótesis a partir de los ejemplares de entrenamiento para luego utilizarla a la hora de aproximar nuevos ejemplares, los métodos basados en instancias no analizan los ejemplares de entrada, tan sólo los almacenan. Es a la hora de aproximar un nuevo ejemplar cuando calcula una aproximación local de la función objetivo basada en aquellos ejemplares del conjunto de entrenamiento más similares a él. Esto conlleva la necesidad de incorporar técnicas de indexado eficiente de los ejemplares de entrenamiento, así como optimizar los cálculos de la medida de *distancia* utilizada para evaluar la *similitud* entre ejemplares.

A continuación se detallará uno de los algoritmos de aprendizaje basado en instancias más ampliamente utilizados: el **modelo de vecinos más cercanos**.

Modelo de vecinos más cercanos

El modelo de vecinos más cercanos asume que cada ejemplar corresponde a un punto en el espacio n-dimensional $\Re^n$.

Para identificar los vecinos más cercanos de un punto, se hace necesaria la definición de

una métrica para medir la distancia entre dos ejemplares $D(e_1, e_2)$. Una de las medidas más utilizadas es la *distancia Euclídea* y se define según la expresión (3.4). Otra alternativa es la *distancia de Manhattan*, donde el valor absoluto de la diferencia entre los valores de los atributos es sumada sin elevarse al cuadrado, como dicta la expresión (3.5). Aunque estas medidas son unas de las más comúnmente utilizadas, según [27] no son apropiadas cuando algunos atributos se miden de forma diferente (como por ejemplo altura y peso), ya que cambiar la escala de una dimensión podría alterar el conjunto de vecinos más cercanos. Para solucionar este problema se propone estandarizar la escala para cada dimensión. Para ello se propone la *distancia de Mahalanobis* [8], la cual mide la desviación estándar de cada atributo sobre el conjunto de los datos y se expresan los valores de un atributo como múltiplos de la desviación estándar de dicho atributo. En el caso de atributos discretos, la medida de distancia más apropiada es la *distancia de Hamming* [27], la cual consiste en sumar el número de atributos en que e_1 y e_2 difieren.

$$D(e_1, e_2) = \sqrt{\sum_{r=1}^{n} (a_r(e_1) - a_r(e_2))^2} \tag{3.4}$$

$$D(e_1, e_2) = \sum_{r=1}^{n} |(a_r(e_1) - a_r(e_2))| \tag{3.5}$$

Una vez definida la medida mediante la cual se especificará el grado de *similitud* entre dos ejemplares cualquiera, queda definir el concepto de *vecindad*. La **vecindad** de un ejemplar e está formada por el conjunto de los k ejemplares $(e_1, e_2, ..., e_k)$ que son más similares a e. Escoger el valor de k es una decisión crucial, ya que con valores demasiado bajos la aproximación se hace a partir de muy pocos ejemplares y el algoritmo es más susceptible a posible ruido en los datos, mientras que con valores de k demasiado altos puede hacerse una aproximación basada en ejemplares que en realidad no tan similares al ejemplar de entrada. Según [27], en la práctica un valor entre 5 y 10 proporciona buenos resultados para la mayoría de los conjuntos de datos con pocas dimensiones.

Cuando la función a aproximar es de naturaleza discreta, el algoritmo devuelve el valor del atributo objetivo más común entre el conjunto de los k vecinos más cercanos. Por otro lado, si la función a aproximar es de naturaleza continua el algoritmo devuelve la media aritmética

de los ejemplares vecinos según la expresión (3.6).

$$f(e_q) = \frac{\sum_{i=1}^{k} f(e_i)}{k} \tag{3.6}$$

Una mejora obvia para el algoritmo de los *k vecinos más cercanos* es ponderar la contribución de cada ejemplar vecino a la aproximación de la función objetivo. Esto consiste en asignar un mayor peso al valor aportado por aquellos vecinos más cercanos al ejemplar de entrada. Dicho peso puede ser determinado según el *ranking* de cercanía o según distancia.

Por último, un aspecto crítico en el diseño de este algoritmo deriva del hecho de que casi todas las operaciones son realizadas a la hora de aproximar nuevos ejemplares de entrada. Esto implica que en situaciones en las que el conjunto de ejemplares de entrenamiento sea muy grande puede hacerse necesaria la utilización de algún método de indexado eficiente. En [32] se propone la utilización de *kd-trees* [10], una estructura de datos que permite la organización de puntos definidos en un espacio de k dimensiones según su proximidad.

3.1.5. Aprendizaje bayesiano

Este tipo de métodos toman todas las posibles hipótesis ofrecidas por los datos, ponderadas por sus respectivas probabilidades, y realizan sus predicciones mediante inferencia probabilística. Esto es una gran diferencia con respecto otros métodos que calculan únicamente la *mejor* hipótesis que es la que se utiliza para realizar las predicciones.

Para este tipo de métodos, la *mejor* hipótesis es aquella que es la más probable. La probabilidad de que una hipótesis h_i se sostenga en una situación donde se ha observado un fenómeno d, o probabilidad **a posteriori** de h_i, $P(h_i|d)$, se calcula aplicando la regla de Bayes (3.7) a partir de cierta información conocida como las probabilidades **a priori** de todas las hipótesis $P(h_i)$ y la **verosimilitud** de los datos dada cada una de las hipótesis, $P(d|h_i)$. La ventaja de este tipo de métodos radica en que cuantos más fenómenos son observados, *la verdadera hipótesis domina la predicción* [27].

$$P(h_i|d) = \frac{P(d|h_i)P(h_i)}{P(d)} \tag{3.7}$$

Un aspecto importante a tener en cuenta a la hora de aplicar aprendizaje bayesiano es que

la verosimilitud de los datos se calcula asumiendo que las observaciones son **i.i.d.**, es decir, *independientes e idénticamente distribuidas* [32].

En problemas reales el espacio de hipótesis suele ser muy grande, por lo que en algunos casos el aprendizaje bayesiano puede no ser factible. En estos casos se debe recurrir a métodos aproximados o simplificados, como basar la predicción únicamente en la hipótesis más probable. Este método se denomina **máximo a posteriori**, o hipótesis **MAP** h_{MAP}. Las predicciones realizadas mediante una hipótesis h_{MAP} son aproximadamente bayesianas. Conforme se observan más datos, la hipótesis h_{MAP} y la bayesiana se van aproximando, ya que las hipótesis competidoras con h_{MAP} se van haciendo cada vez menos probables. Otra simplificación más puede hacerse en situaciones en las que no hay razón alguna para preferir una hipótesis sobre otra. En estos casos se asume una distribución uniforme a priori sobre el espacio de hipótesis y el aprendizaje se reduce a elegir una hipótesis que maximice $P(d|h_i)$. Esta hipótesis recibe el nombre de **hipótesis de máxima verosimilitud**, h_{ML}. Esta simplificación funciona bien cuando el conjunto de datos es muy grande, ya que los datos imperan sobre la distribución a priori de las hipótesis, pero presenta problemas con conjuntos de datos pequeños.

Clasificador Naive Bayes

El clasificador **naive bayes** es un modelo concreto de red bayesiana [23] muy comúnmente utilizado en problemas de clasificación como *benchmark*, ya que ha sido demostrado que proporciona una eficiencia comparable a las redes neuronales o los árboles de decisión en ciertos tipos de problemas [32]. Es capaz de manejar datos con atributos continuos o discretos, aunque el atributo objetivo (o clase) debe ser de naturaleza discreta. El adjetivo *naive* significa *ingenuo*, ya que el algoritmo asume que los valores de los atributos son condicionalmente independientes entre sí dada la clase. Esto significa que se asume que, dada la clase de un ejemplar C, la probabilidad de observar una serie de valores de atributos $(a_1, a_2, ..., a_n)$ es el producto de las probabilidades de cada uno de los atributos según la expresión (3.8).

$$P(a_1, a_2, ..., a_n|C) = \prod_i P(a_i|C) \tag{3.8}$$

A partir de las probabilidades a priori y la verosimilitud de los datos de entrenamiento, el clasificador *naive* bayes predice la clasificación de un nuevo ejemplar mediante inferencia probabilística, según la expresión (3.9).

$$P(C|a_1, a_2, ..., a_n) = \frac{P(C)\prod_{i=1}^{n} P(a_i|C)}{\prod_{i=1}^{n} P(a_i)} \tag{3.9}$$

3.1.6. Algoritmos Genéticos

Los algoritmos genéticos (AG) [22] son un método de aprendizaje y búsqueda basado en la teoría de la evolución biológica propuesta por Darwin. Este método se caracteriza porque en lugar de buscar hipótesis más específicas a partir de las más generales, o viceversa, genera hipótesis desconocidas sucesoras a partir de la combinación y mutación de las mejores hipótesis conocidas. La calidad de las hipótesis es medida por una función de **idoneidad** que otorga valores numéricos mayores a las hipótesis de mayor calidad. La *mejor* hipótesis será por tanto aquella que maximice el valor de idoneidad. Para poder generar nuevas hipótesis a partir de las ya conocidas, cada hipótesis debe ser representada como una cadena sobre un alfabeto finito. Un ejemplo de esta representación es una cadena de ceros y unos.

Aunque existen infinidad de implementaciones diferentes de algoritmos genéticos, todas ellas varían en pequeños detalles y comparten la estructura general presentada en esta sección.

El procedimiento básico de todo AG opera iterativamente sobre un conjunto de hipótesis llamado **población**. En cada iteración el algoritmo selecciona hipótesis o **individuos** de la población, los cuales son combinados entre ellos para generar nuevos individuos descendientes. Esta operación de combinación se denomina **cruce**. Cada nuevo individuo generado puede ser sometido a una operación de **mutación** según una pequeña probabilidad independiente. Antes de finalizar cada iteración, cada individuo generado es añadido a la población inicial, favoreciendo así la posibilidad de que los descendientes de mayor calidad puedan ser escogidos en nuevas iteraciones para generar nuevos individuos. El algoritmo finaliza cuando algún individuo de la población ha superado un determinado **umbral de idoneidad** o cuando se han realizado un número de iteraciones dado.

La selección de los individuos de la operación de cruce se realiza de forma probabilística,

Algoritmo 2 AlgoritmoGenetico($Poblacion$, $Idoneidad$)

repeat
  $nueva\_poblacion \leftarrow$ conjunto vacío
  for $i = 1$ hasta Tamaño($Poblacion$) **do**
    $x \leftarrow$ SeleccionProbabilistica($Poblacion$, $Idoneidad$)
    $y \leftarrow$ SeleccionProbabilistica($Poblacion$, $Idoneidad$)
    $hijo \leftarrow$ Reproducir(x, y)
    if probabilidad aleatoria pequeña **then**
      $hijo \leftarrow$ Mutar($hijo$)
    end if
    añadir $hijo$ a $nueva\_poblacion$
  end for
  $Poblacion \leftarrow Poblacion + nueva\_poblacion$
until algún individuo de $Poblacion$ es bastante adecuado o ha pasado bastante tiempo
return el mejor individuo de $Poblacion$ según $Idoneidad$

según la expresión (3.10). Así los individuos con mayor calidad tienen una mayor probabilidad de ser elegidos para generar individuos. Al igual que ocurre en la evolución biológica, los individuos de buena calidad suelen ser generados a partir de individuos de buena calidad.

$$Prob(h_i) = \frac{Idoneidad(h_i)}{\sum_{j=1}^{p} Idoneidad(h_j)} \tag{3.10}$$

En la operación de cruce los individuos seleccionados son combinados para dar lugar a nuevos individuos. Esta combinación se realiza seleccionando aleatoriamente un punto de cruce en las posiciones de la cadena. El nuevo individuo surgirá a raíz de combinar la parte anterior al punto de cruce de la cadena de un individuo con la parte posterior al punto de cruce de la cadena del otro.

Algoritmo 3 Reproducir(x, y)

$n \leftarrow$ Longitud(x)
$c \leftarrow$ número aleatorio de 0 a $n - 1$
return Concatenar(Subcadena(x, 0, c), Subcadena(y, $c + 1$, $n - 1$))

Cada individuo generado tiene una pequeña probabilidad independiente de ser sometido a la operación de mutación. Esta operación consiste en seleccionar aleatoriamente una posición de la cadena y modificar de forma aleatoria el valor en dicha posición. El objetivo de esta operación es identificar posibles hipótesis de mayor calidad muy similares a las obtenidas

mediante la operación de cruce. Las hipótesis resultado de la operación de mutación serían difícilmente alcanzables tan sólo con la operación de cruce.

La figura 3.3 muestra los individuos seleccionados para la operación de cruce en (a). Cada pareja de individuos es combinada de acuerdo al punto de cruce obteniendo así los individuos generados en (b). Por último, algunos de los nuevos individuos son sometidos a la operación de mutación en (c).

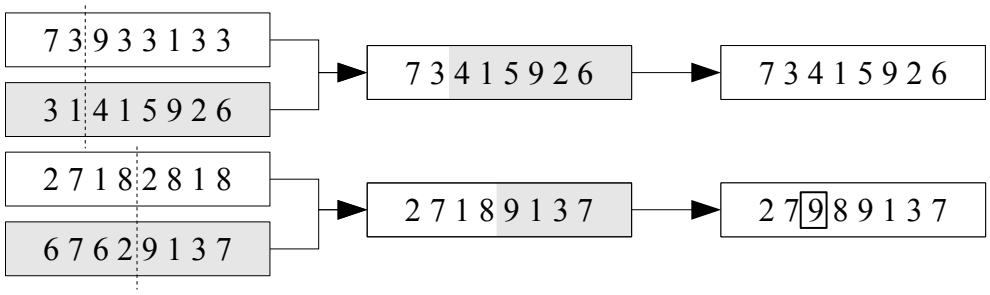


Figura 3.3: Proceso de cruce y mutación en algoritmos genéticos

Una extensión al modelo básico de AG es la incorporación de **esquemas** [27]. Un esquema es una cadena de representación de un individuo en la que se *bloquean* ciertos elementos y se permite la modificación del resto. Por ejemplo, el esquema 101***** describe todas aquellas hipótesis cuyos primeros elementos sean 101. Estas hipótesis se llaman **instancias** del esquema. La operación de cruce está restringida a los elementos no fijos. De esta forma, si la calidad del esquema es superior a la media, todas sus instancias tendrán también una calidad superior a la media.

3.1.7. Otros métodos

Los métodos vistos hasta ahora en esta sección son unos de los más populares en el campo del aprendizaje máquina y unos de los más utilizados en procesos de minería de datos. No obstante existen muchos más, algunos de los cuales serán introducidos brevemente a continuación. Los siguientes métodos, aunque de gran utilidad en ciertas situaciones, no han sido

utilizados en el desarrollo del presente proyecto. Por este motivo son incluidos en esta sección.

Redes Bayesianas

Las redes bayesianas son una representación gráfica de dependencias para razonamiento probabilístico, en la cual los nodos representan variables aleatorias y los arcos las relaciones de dependencia directa entre las variables. Dado este modelo, es posible estimar la probabilidad a posteriori de las variables no conocidas a partir de las probabilidades a priori de las variables conocidas.

La inferencia probabilística en este tipo de modelos se realiza propagando los efectos de las variables conocidas a través de la red para calcular la probabilidad a posteriori de las variables de interés.

Existen diferentes tipos de algoritmos de propagación de probabilidades, los cuales dependen de la naturaleza del grafo modelado y de si se pretende calcular tan sólo el valor de una única variable o de un conjunto de ellas simultáneamente. Los principales tipos de algoritmo de inferencia son [28]:

- En casos donde se pretenda calcular una única variable, el **algoritmo de eliminación** (o *variable elimination*) es el más utilizado.
- Si se han de calcular varias variables de forma simultánea en grafos sencillos se suele utilizar el algoritmo de propagación de **Pearl**.
- Si se han de calcular varias variables simultáneamente en grafos de cualquier estructura se puede optar por: algoritmos de **agrupamiento**, **simulación estocástica** y **condicionamiento**.

Redes Neuronales

Las redes neuronales son un tipo de modelo inspirado en el modelo biológico de sistema de neuronas interconectadas entre sí. Una neurona es una célula del cerebro cuya función principal es la recogida, procesamiento y emisión de señales eléctricas. Es comúnmente aceptada la creencia de que la capacidad de procesamiento de información del cerebro proviene

principalmente de redes de este tipo de neuronas. Por este motivo, algunos de los primeros trabajos en el campo de la Inteligencia Artificial trataban de crear redes de neuronas artificiales para diseñar sistemas capaces de aprender de un dominio determinado y actuar en función de dicho aprendizaje.

Las redes neuronales están compuestas de nodos o unidades conectadas a través de conexiones dirigidas. Una conexión de la unidad j a la unidad i sirve para propagar la activación a_j de j a i. Además cada conexión tiene un **peso** numérico $W_{j,i}$ asociado, que determina la importancia y el signo de la conexión.

Cada nodo i de la red calcula primero una suma ponderada de sus entradas:

$$in_i = \sum_{j=0}^{n} W_{j,i} a_j \tag{3.11}$$

Luego se aplica una **función de activación** g al resultado de la expresión (3.11) para producir una salida:

$$a_i = g(in_i) = g(\sum_{j=0}^{n} W_{j,i} a_j) \tag{3.12}$$

La función de activación g se diseña con dos objetivos. El primero es que es deseable que la unidad esté *activa* (cercana a $+1$) cuando se proporcionen las entradas correctas e inactiva (cercana a 0) cuando las entradas sean *erróneas*. El segundo objetivo es que la activación debe ser no *lineal*, ya que en otro caso la red neuronal en su totalidad se colapsaría con una sencilla función lineal. Dos de las funciones g más comunes son la función **umbral** y la **sigmoide** o **logística** [27].

Existen dos categorías principales de redes neuronales atendiendo a su estructura: acíclicas o **redes de alimentación hacia delante** y cíclicas o **redes recurrentes**. Una red con alimentación hacia delante representa una función de sus entradas actuales, por lo que el único estado interno es el representado por sus pesos. Las redes recurrentes por otro lado permiten que sus salidas alimenten sus propias entradas. Esto hace posible que este tipo de redes dispongan de una *memoria* a corto plazo [23].

Análisis de clusters (Clustering)

El análisis de clusters o *Clustering* es una colección de métodos estadísticos que permiten agrupar ejemplares descritos mediante una colección (generalmente bastante grande) de atributos. Estas técnicas son un caso particular de método de aprendizaje basado en instancias, aunque en problemas de clustering los ejemplares no pertenecen a ninguna clasificación determinada. El agrupamiento de estos se hace por tanto en función de la similitud de las descripciones de dichos ejemplares. Al igual que en otros métodos de aprendizaje basado en instancias, los métodos de clustering tratan cada ejemplar como un punto en un espacio n-dimensional, donde n es el número de atributos que describen dicho ejemplar.

El agrupamiento de instancias puede ser exclusivo, con solapamiento o probabilístico. En agrupamiento exclusivo cada instancia puede pertenecer a un único grupo a la vez, en el agrupamiento con solapamiento puede haber instancias que pertenezcan a más de un grupo a la vez y en el agrupamiento probabilístico cada instancia pertenece a cada grupo con una probabilidad determinada.

El algoritmo clásico para agrupar instancias en un problema de clustering se denomina k**-means**. El primer paso en este algoritmo es especificar el valor de k, que indica el número de grupos en los que se van a agrupar las instancias. A partir del valor de k, el algoritmo elige al azar k instancias que serán los puntos centrales de los grupos que son creados. Una vez definidos los k grupos en función de sus puntos centrales, el resto de instancias son asignadas al grupo correspondiente al punto central con el que cada una tenga menor distancia. La medida de distancia escogida normalmente es la distancia *Euclídea*, tal y como aparece en la expresión (3.4). Cuando todas las instancias han sido asignadas a algún grupo, se recalcula el centro de cada grupo y el proceso completo se vuelve a repetir una iteración más. El algoritmo termina cuando los grupos obtenidos en una iteración y en la siguiente son idénticos.

3.1.8. Evaluación de algoritmos de aprendizaje

Un problema determinado puede ser abordado con distintos métodos de aprendizaje. Unos funcionarán mejor que otros según las características específicas del problema en cuestión. Por lo tanto, se hacen necesarias formas sistemáticas de evaluar cómo de bien o mal funciona

un algoritmo determinado en comparación con otro u otros.

Según sean las características específicas del problema a tratar, se utilizará una medida u otra para evaluar las hipótesis disponibles. Por ejemplo, en problemas de clasificación el interés se centra en encontrar aquella hipótesis que clasifique correctamente un mayor número de ejemplares, mientras que en problemas de regresión se trata de encontrar aquella hipótesis que produzca el menor error medio al cuadrado en el conjunto de predicciones realizadas sobre los ejemplares de test.

Una hipótesis no debe ser escogida nunca basándose en la proporción de predicciones correctas realizadas sobre el mismo conjunto de datos que ha sido utilizado para inducirla. En estos casos el indicador de predicciones correctas será una medida muy optimista de la eficiencia de la hipótesis, a la hora de generalizar para predecir nuevos ejemplares en el futuro. Cuando una hipótesis es inducida y evaluada con los mismos datos se dice que hay un problema de **peeking**. Pero el interés no se centra en la predicción correcta de ejemplares ya conocidos, sino en la predicción de ejemplares futuros cuyo valor de la función objetivo es totalmente desconocida.

La aproximación básica para evaluar una hipótesis dada es por tanto dividir el conjunto de datos disponible en dos subconjuntos: un conjunto de datos utilizado para entrenar el algoritmo (conjunto de **entrenamiento**) y otro para validar su comportamiento en la predicción de nuevos ejemplares (conjunto de **test**). Ambos conjuntos deben ser muestras representativas del problema a tratar. Esto evita el riesgo de escoger una hipótesis h que produce unos mejores resultados en el conjunto de entrenamiento que otra h', pero que a la hora de generalizar para predecir ejemplares futuros h' da mejores resultados que h. Este es el problema de *sobreajuste* que se mencionaba anteriormente en esta sección. Además, de esta forma también se suprime el riesgo de *peeking*.

Una vez escogida una hipótesis inducida con un conjunto de datos de entrenamiento y validada con un conjunto independiente de datos de test, normalmente, los datos de test se suelen fusionar con los datos de entrenamiento y se vuelve a inducir la hipótesis. Esto es una forma de maximizar el conjunto de datos de entrenamiento a partir de los cuales la hipótesis final a utilizar será inducida. Además no supone ningún problema, ya que como se asume que ambos conjuntos son muestras representativas del problema a tratar, el error cometido en el

conjunto de datos de test será un buen indicador del error que será cometido en la predicción de ejemplares futuros.

La situación ideal para evaluar una hipótesis es disponer de un conjunto de datos muy grande. Así es posible obtener un gran conjunto de datos de entrenamiento con el que inducir una hipótesis lo más aproximada posible a la hipótesis real que describe el problema. Del mismo modo es posible obtener un gran conjunto de datos de test con el que aproximar en el mayor grado posible el error cometido ante ejemplares futuros desconocidos. El problema viene cuando no se dispone de un conjunto de datos demasiado grande. En esta situación el problema consiste en establecer el tamaño de cada conjunto encontrando un equilibrio entre ambos grados de aproximación: el de la hipótesis inducida y el del error cometido en ejemplares futuros.

Validación cruzada

En situaciones donde el conjunto de datos disponible para inducir y evaluar una hipótesis es reducido, lo normal es tomar alrededor de $\frac{2}{3}$ del conjunto total de los datos para inducir la hipótesis y el $\frac{1}{3}$ restante para validarla. Si ambos subconjuntos son muestras representativas del problema en cuestión, obtendremos un grado de aproximación más o menos preciso tanto de la hipótesis inducida, como del error de predicción cometido. Sin embargo, en la práctica no hay forma de saber si una muestra de datos es representativa o no y, aunque la mayoría de los datos disponibles formen una muestra bastante representativa, en la elección del conjunto de datos de test podemos incluir ejemplares que no lo son, o viceversa. Este problema se soluciona en buen grado con el método de **validación cruzada**.

Este método consiste en repetir los procesos de inducción y validación de una hipótesis repetidas veces escogiendo en cada iteración un subconjunto de datos de entrenamiento y test diferentes. Esto proporciona una mayor probabilidad de que existan muestras representativas en ambos conjuntos de datos. Una vez finalizado este proceso iterativo, el error de aproximación final es calculado según la media aritmética de los errores cometidos en cada iteración.

Más concretamente, en un proceso de validación cruzada de k hojas consiste en dividir el conjunto total de datos en $\frac{N}{k}$ subconjuntos, donde N es el número total de ejemplares disponibles. Una vez hecho esto, se emplean $k-1$ subconjuntos en inducir una hipótesis y el

subconjunto restante para validarla. Este proceso se repite k veces, y el grado de error estimado final es calculado según la media aritmética de los errores cometidos en cada iteración.

Escoger el valor de k es otro problema que hay que solucionar a la hora de utilizar el método de validación cruzada. Según [32], a raíz de varios experimentos realizados se ha demostrado que un valor de $k = 10$ funciona forma bastante aproximada a la óptima en los tipos de problemas más comunes. A esta forma particular de validación cruzada se le denomina **validación cruzada de diez hojas** y es un estándar en la validación de algoritmos de aprendizaje.

Otra forma particular de validación cruzada es el método de **validación cruzada dejando uno fuera** (*leave-one-out cross-validation*). En esta forma particular $k = N$, lo que significa que se realizan N iteraciones y en cada una de ellas se utiliza un conjunto de $N - 1$ ejemplares para inducir una hipótesis que será validada utilizando un sólo ejemplar. Después de repetir este proceso N veces, se cuentan los errores cometidos en cada iteración y esta será la aproximación del error devuelta por el método.

3.2. Síntesis de Imagen Realista

3.2.1. Introducción

Desde la época de las primeras pinturas rupestres en el paleolítico, el hombre ha tenido la necesidad de construir imágenes bidimensionales que representen un mundo real o imaginario tridimensional. No fue hasta el renacimiento en el siglo XVI cuando Albrecht Dürer alcanzó un importante nivel de fotorrealismo gracias a sus revolucionarios estudios de proyección y perspectiva.

Estas ideas desarrolladas por el famoso pintor alemán fueron mejoradas por otros artistas posteriores como Vermeer y Rembrandt y adaptadas a la informática por primera vez por Arthur Appel en 1968 y mejoradas por Bouknight en 1970 con el método de render *Scanline* [11]. En estos métodos la rejilla que empleaba Dürer se reemplaza por un plano de imagen en el que cada pixel se corresponde con una cuadrícula de la rejilla.

El estudio de la perspectiva mediante "trazado de cuerdas" fue estudiado en 1637 por Descartes en los primeros tratados de óptica geométrica, donde se explicaba la forma del arco iris. Esta primera aproximación al trazado de rayos fue, en esencia, la misma en la que se basó Whitted en 1980 para formular el método recursivo de trazado de rayos [31]. En este método se basan la mayoría de los algoritmos de renderizado realistas que se emplean en la actualidad.

En los últimos 15 años, los gráficos por computador han pasado de formar parte de la comunidad científica a invadir el mercado cinematográfico, los videojuegos, la visualización médica y los sistemas de diseño asistido por computador. Una de las líneas de trabajo que más repercusión han tenido en todas estas áreas es la de representar imágenes de objetos tridimensionales, cuya descripción matemática está almacenada en la memoria de un computador, con el mayor realismo posible de forma que un observador humano no pueda diferenciar si se trata de una fotografía o de una imagen generada mediante un motor de render. Este campo de trabajo se ha denominado **síntesis de imagen realista**.

En la construcción de cualquier imagen interviene un proceso de selección, abstracción y aproximación de las propiedades del objeto a representar. Como señala Ferwerda [14], podemos identificar tres niveles de realismo en gráficos por computador que se diferencian

en la aproximación, abstracción u omisión de las propiedades de los objetos que forman la escena a representar:

- **Realismo físico**: La imagen proporciona la misma *estimulación visual* que la escena a representar. Esta definición implica que la imagen debe tener la misma representación de irradiancia que la escena para cada punto de vista seleccionado. Así, el modelo matemático debe ser completo y preciso, conteniendo todos los datos referentes a geometría, materiales, iluminación, etc. y el motor de render debe ser capaz de simular de forma precisa la interacción de la luz en este entorno virtual. Este nivel de realismo es difícilmente alcanzable debido a que en la mayoría de los casos, los dispositivos de visualización actuales no soportan imágenes de alto rango dinámico, y los métodos son computacionalmente intensivos. Además, el sistema de visión humano no es capaz de percibir toda la información que estos métodos de simulación pueden aportar.

- **Fotorrealismo**: La imagen generada debe proporcionar la misma *respuesta visual* que la escena a representar, aunque la energía física que proviene de la imagen pueda ser diferente que la percibida en la escena. De esta forma es posible tomar en cuenta las limitaciones del sistema de visión humano para simplificar el proceso de generación de imágenes por computador (como por ejemplo, describir los colores mediante una tripleta de colores RGB o CMY). Teniendo en cuenta las limitaciones del sistema de visión humano se han realizado algunos trabajos que tratan de simplificar el cálculo del render en animaciones, como la aproximación de Myszkowski [24] que estudia la coherencia de la distribución de la luz en mapeado de fotones. Pese a que conseguir imágenes fotorrealistas es menos costoso que sus equivalentes físicamente realistas, sigue siendo un proceso muy costoso computacionalmente y se continuan estudiando alternativas de optimización.

- **Realismo funcional**: La imagen generada debe contener la misma *información visual* que la escena a representar. En este caso *información* se refiere al conocimiento sobre las propiedades de tamaño, forma, posición y materiales. En muchos casos el realismo funcional es preferible al fotorrealismo, como en el caso de ilustraciones técnicas, mapas, etc. En algunos casos donde el fotorrealismo podría ser adecuado, resulta más

conveniente el realismo funcional por ser computacionalmente más sencillo, como en el caso de los simuladores de vuelo; pese a que las imágenes no son fotorrealistas, el piloto es capaz de obtener la misma información visual del simulador que del entorno real de vuelo.

Para la construcción de imágenes fotorrealistas es imprescindible que, además de la perspectiva, el artista comprenda y logre simular el comportamiento de la luz y las características del sistema visual humano. Desde los inicios del estudio de la óptica los físicos han desarrollado modelos matemáticos para estudiar la interacción de la luz en las superficies. Con la aparición del microprocesador, los ordenadores tuvieron suficiente potencia como para poder simular estas complejas interacciones. Así, empleando un ordenador y partiendo de las propiedades geométricas y de materiales especificadas numéricamente es posible simular la reflexión y propagación de la luz en una escena. A mayor precisión en esta simulación, mayor nivel de realismo conseguiremos en la imagen resultado.

A pesar de que el objetivo parece sencillo, existen algunos problemas y limitaciones importantes. El primero es debido al excesivo coste computacional de los métodos de cálculo más realistas. Para cualquier proyecto real de obtención de una imagen, hay que establecer un límite en el nivel de detalle de la escena que queremos simular, indicados en términos del número de interacciones máximas de la luz con las superficies, resolución espacial de la imagen, etc. Debida a esta primera restricción, resulta crítico el uso de algoritmos y métodos que incorporen conocimiento para la ayuda a la evaluación de una escena. Dicha evaluación permite decidir en qué zonas se va a dedicar el mayor esfuerzo computacional. Otro importante problema se debe a la naturaleza discreta de los dispositivos de visualización. La necesidad de representar un espacio continuo (en términos de geometría, colores, dimensión temporal, etc.) en un medio discreto (un monitor *raster* cuenta con una determinada resolución, profundidad de color, tasa de refresco, etc.) hace necesario el uso de técnicas para minimizar el error cometido (utilizando mecanismos de antialiasing, corrección de gamma, motion blur, etc.).

Esta conexión entre la simulación del comportamiento de la luz y el nivel de realismo queda patente en las aproximaciones propuestas en diferentes métodos de render que serán expuestos en detalle en 3.2.3.

A un alto nivel de abstracción, podemos realizar una primera taxonomía de métodos de

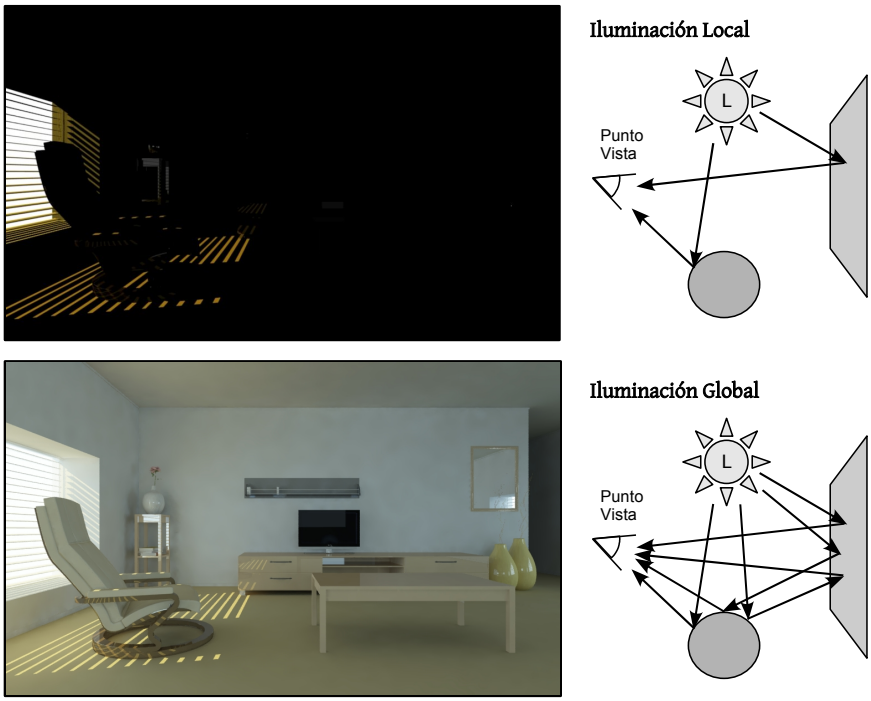


Figura 3.4: Una misma escena renderizada con dos métodos diferentes. **Arriba**: Iluminación Local (Scanline). **Abajo**: Iluminación Global (Mapeado de Fotones).

render entre aquellos que realizan una simulación de **iluminación local**, teniendo en cuenta únicamente una interacción de la luz con las superficies, o los métodos de **iluminación global** que tratan de calcular *todas*[2] las interacciones de la luz con las superficies de la escena. En la figura 3.4 se muestra el resultado de renderizar la misma escena con un método de iluminación local y uno global. Los modelos de iluminación global incorporan la *iluminación directa* que proviene de la primera interacción de las superficies con las fuentes de luz, así como la *iluminación indirecta* reflejada por otras superficies existentes en la escena.

[2]Debido a que es imposible calcular las infinitas interacciones de los rayos de luz con todos los objetos de la escena, las aproximaciones de iluminación global se ocuparán de calcular algunas de estas interacciones, tratando de minimizar el error de muestreo.

3.2.2. La Ecuación de Render

Tras la aparición de los primeros métodos de *Iluminación Global*, Kajiya describió en [20] la ecuación general de Render. En su artículo además propuso el empleo de integración basada en Monte Carlo para su resolución (la base de los métodos de PathTracing). De forma simplificada, la ecuación de Render puede expresarse como:

$$L_o(\vec{x}, \vec{\omega}) = L_e(\vec{x}, \vec{\omega}) + L_r(\vec{x}, \vec{\omega}) \tag{3.13}$$

que indica que la radiancia de salida de una superficie L_o (en adelante la denotaremos simplemente con L) puede calcularse como la suma de la radiancia emitida L_e y la radiancia reflejada L_r. Una de las leyes básicas de la geometría óptica emplea la suposición de que la radiancia no cambia con la propagación de la luz (suponiendo que no hay absorción por parte de las superficies) y en el espacio la luz viaja en línea recta.

De esta forma, suponiendo que un punto $\vec{x}$ de una superficie receptora ve un punto $\vec{x}'$ de una superficie emisora en la dirección $\vec{\omega}$, la radiancia de entrada es igual a la radiancia de salida:

$$L_i(\vec{x}, \vec{\omega_i}(\vec{x}', \vec{x})) = L_o(\vec{x}', \vec{\omega_o}'(\vec{x}, \vec{x}''))$$

La expresión de la ecuación de reflexión L_r de la ecuación (3.13) se define según [20] como:

$$L_r(\vec{x}, \vec{\omega}_r) = \int_{\Omega_i} f_r(\vec{x}, \vec{\omega}_i \rightarrow \vec{\omega}_r)\ L_i(\vec{x}, \vec{\omega}_i)\ cos\theta_i d\omega_i \tag{3.14}$$

que se calcula integrando la radiancia de entrada sobre una semiesfera situada con centro en $\vec{x}$ y orientada de forma que el *polo norte* de la semiesfera esté alineado con el vector normal de la superficie.

De forma general podemos decir que a mayor simplificación en la resolución de los términos de esta ecuación tendremos métodos menos realistas (y computacionalmente menos costosos).

3.2.3. Métodos de Render

A un alto nivel de abstracción podemos ver el proceso de *render* como el encargado de convertir la descripción de una escena tridimensional en una imagen típicamente *raster*

bidimensional. En los primeros años de estudio de esta disciplina, la investigación se centró en cómo resolver problemas relacionados con la detección de superficies visibles, sombreado básico, etc. Según se encontraban soluciones a estos problemas, se continuó el estudio de algoritmos más precisos que simularan el comportamiento de la luz de una forma más fiel.

A continuación se presentan las principales técnicas de render ordenadas cronológicamente. La mayoría de estas técnicas cuentan con versiones híbridas, versiones ampliadas y optimizaciones para ciertos dominios de aplicación. Sin embargo, la explicación en este punto se centrará en las versiones originales de cada método.

RayCasting

Gracias al decremento en el precio de la memoria de los ordenadores en la segunda mitad de los años 60, los gráficos *raster* (o de mapas de bits) ganaron popularidad frente a los vectoriales. Los primeros métodos de sombreado de superficies propuestos por Gouraud y Phong no realizaban ninguna simulación física de la reflexión de la luz, calculando únicamente las contribuciones locales de iluminación, tomando en cuenta la posición de la luz, el observador y el vector normal de la superficie.

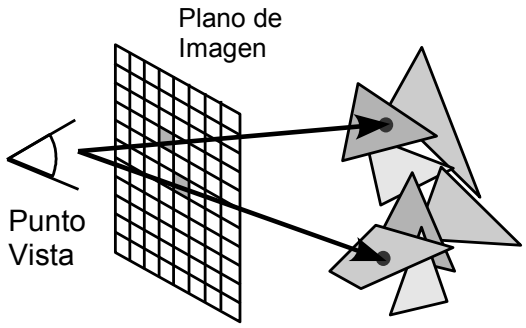


Figura 3.5: Método de RayCasting

En 1968 Arthur Appel [9] describió el primer método para generar imágenes por computador lanzando rayos desde el punto de vista del observador. En su trabajo generaba la imagen

resultado en un plotter donde dibujaba punto a punto el resultado del proceso de render. La idea general del método de *RayCasting* es lanzar rayos desde el plano de imagen, uno por cada píxel, y encontrar el punto de intersección más cercano con los objetos de la escena. La principal ventaja de este método frente a los métodos de tipo *scanline* que emplean *zbuffer* es que es posible generar de forma consistente la imagen que represente el mundo 3D ya que cualquier objeto que pueda ser descrito mediante una ecuación puede ser representado de forma correcta mediante *RayCasting*.

RayTracing

El método de trazado de rayos (propuesto por Whitted [31] como mejora del método de RayCasting [9]) es un método elegante y sencillo que permite calcular de una forma unificada la reflexión y la refracción de la luz, sombras, eliminación de superficies ocultas y otros efectos necesarios para conseguir escenas fotorrealistas. De este método surgieron aproximaciones más completas que resolvían de forma más exacta la ecuación de render, basándose en los mismos principios de trazado de los caminos que sigue la luz. Por esta razón, prestaremos especial atención a la descripción de este método.

La idea básica del trazado de rayos es seguir el camino de la luz desde las fuentes emisoras de fotones hasta que llegan a la posición del observador. La simulación del camino natural de la luz presenta el principal inconveniente de que la mayoría de los rayos nunca llegan al observador (o plano de imagen), lo que resulta computacionalmente prohibitivo. Este es el esquema de trazado de rayos hacia delante (*forward RayTracing*.

Para evitar trazar un número alto de rayos que no llegarán al plano imagen, y como solución sencilla a la discretización del espacio continuo en píxeles, se emplea el trazado de rayos hacia atrás (*backward RayTracing*), donde los rayos parten del plano imagen hasta alcanzar los objetos de la escena. La figura 3.6 muestra un esquema general de los componentes básicos que forman un trazador de rayos hacia atrás.

Una de las principales diferencias entre el método de RayTracing y el de RayCasting es el cálculo recursivo de la contribución de la luz debido a la reflexión y refracción que se produce en ciertas superficies. Así, definimos cuatro tipos de rayos:

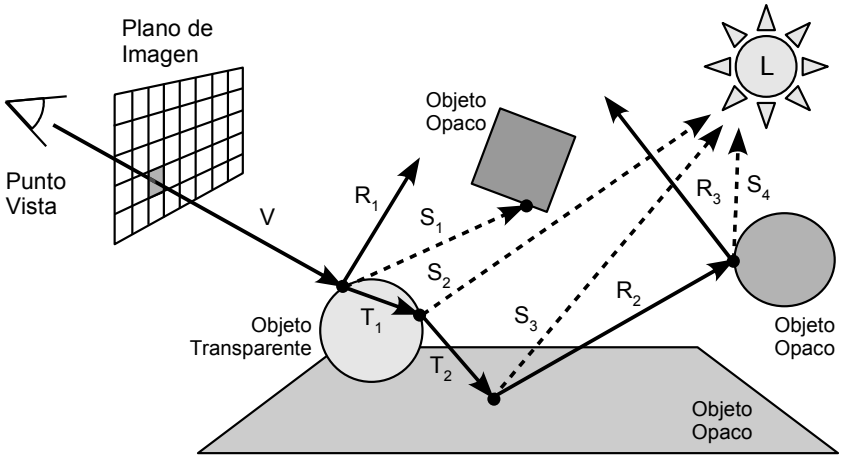


Figura 3.6: Tipos que rayos que intervienen en una solución de RayTracing.

Algoritmo 4 RenderImagen()

for all pixel de la pantalla **do**
  $rayo \leftarrow$ Generar un rayo con origen la cámara y destino el $pixel$
  $color \leftarrow$ RayTracing($rayo$)
  Dibujar $pixel$ con el color $color$
end for

- **Rayos Primarios o Visuales**: Son los rayos que parten de la cámara virtual, pasando por cada uno de los píxeles en el plano de imagen. Para cada elemento de la escena se comprueba si el rayo visual intersecta con alguno de ellos, quedándonos con el punto de intersección más cercano de toda la lista de objetos.

- **Rayos de Sombra**: Parten del punto de intersección con el objeto y tienen dirección hacia las fuentes de luz. De nuevo se realiza una prueba de intersección del rayo con todos los objetos de la escena para ver si hay algún objeto que corte su trayectoria, en cuyo caso el punto de origen del rayo estaría en sombra.

- **Rayos Reflejados**: Si el objeto donde intersectó el rayo tiene propiedades de reflexión de tipo espejo, se generará un nuevo rayo reflejado en ese punto. Este rayo se construirá típicamente en un procedimiento recursivo, pasando a comportarse como un rayo primario en la siguiente iteración del algoritmo.

- **Rayo Transmitidos**: En el caso de objetos en mayor o menor grado transparentes, y de forma análoga al tratamiento para los rayos reflejados, se generará un rayo transmitido. De igual forma, este nuevo rayo se comportará como un rayo primario en la siguiente iteración del algoritmo.

Los algoritmos 4 y 5 muestran el pseudocódigo del algoritmo básico del método de trazado de rayos:

El algoritmo básico se ha dividido en dos funciones principales. En la primera (algoritmo 4) el objetivo es generar los rayos visuales desde el punto de vista de la cámara, teniendo en cuenta las características del tipo de lente utilizada y la resolución de la imagen. En la descripción del rayo se emplea un punto de origen O_r y una dirección d_r, en la forma $R(t) = (O_r + t \cdot d_r)$. Para cada rayo generado, se llama a la función recursiva *RayTracing* (algoritmo 5), pasándole como primer parámetro el rayo visual de la cámara, y como segundo el nivel de recursión inicial. El nivel de recursión es necesario para establecer una condición de parada del algoritmo, y limitará el número de rebotes de los rayos de luz en la escena. Esta función termina devolviendo el color calculado al que se han añadido el resultado de los rayos de sombra, reflexión y refracción.

El trazado de rayos es un método muy costoso computacionalmente, principalmente debido a la necesidad de calcular la intersección más cercana de cada rayo con los objetos que hay en la escena. Esta comparación con cada objeto puede evitarse empleando diversos métodos, como por ejemplo ciertas estructuras de datos que agrupen los objetos de tal forma que comparemos la intersección del rayo con los elementos de esa estructura de datos (típicamente un árbol) y sólo realizaremos la comparación rayo/objeto cuando alcancemos el último nivel de la estructura de datos. En [15] se analizan los criterios para evaluar las técnicas de aceleración en RayTracing según:

- **Aplicabilidad**: La técnica será mejor si es posible aplicarla a todo tipo de rayos, soporta primitivas de geometría sólida constructiva (no únicamente representación de contorno) y se puede aplicar cuando se añade dimensión temporal al trazador.

- **Rendimiento**: Se estudia el rendimiento con escenas muy complejas, analizando si el coste de construir la estructura de datos es menor que el beneficio obtenido de su

Algoritmo 5 RayTracing($rayo$, $prof$)

```
obj ← NULL
dist ← INFINITO
color ← color de fondo
for all objeto en escena do
    intDist ← distancia de intersección más cercana de rayo a objeto
    if intDist < dist then
        dist ← intDist
        obj ← objeto
    end if
    if obj != NULL then
        P ← punto de intersección de rayo con obj
        color ← color de fondo
    end if
    for all fuente de luz en la escena do
        Crear rayo de sombra S con origen fuente
        for all objeto en escena do
            if existe intersección de S con objeto then
                intensidad ← Intensidad(fuente) * Alpha(objeto)
            else
                intensidad ← Intensidad(fuente)
            end if
        end for
        color ← color+ CalcularSombreado(intensidad, P)
    end for
    if prof < PROFUNDIDAD TRAZADO MAXIMA then
        if obj tiene propiedad de reflexión then
            Calcular rayo reflejado Ref
            color ← color+ (RayTracing(Ref, prof + 1) * obj.refl)
        end if
        if obj tiene propiedad de refracción then
            Calcular rayo transmitido Trans
            color ← color+ (RayTracing(Trans, prof + 1) * obj.trans)
        end if
    end if
end for
return color
```

utilización.

- **Recursos**: Estudia los recursos necesarios de memoria, espacio de almacenamiento y número de procesadores necesarios en el computador que quiera hacer uso de esa técnica de optimización.

- **Simplicidad**: En este criterio se tiene en cuenta la dificultad de implementación del algoritmo, su dependencia con alguna arquitectura hardware concreta y si requiere re-escribir mucho código para su utilización.

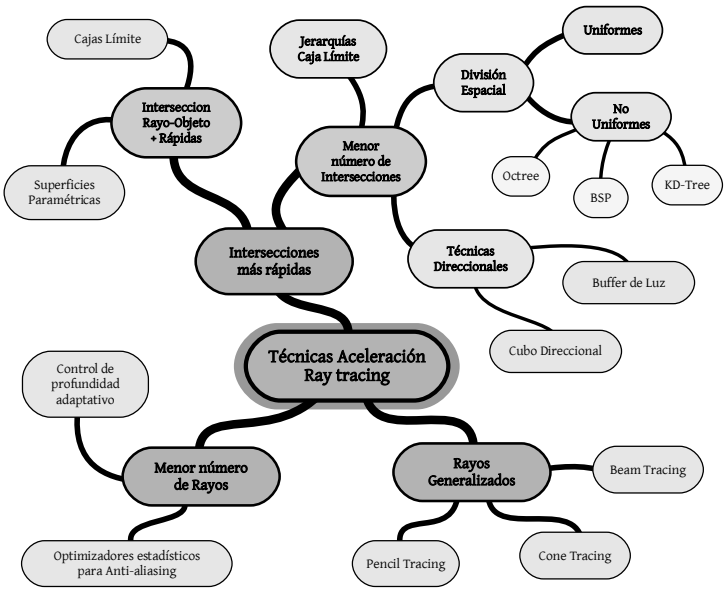


Figura 3.7: Taxonomía de clasificación para las técnicas de aceleración de cálculos en trazado de rayos.

Existen varias opciones para la optimización de un trazador de rayos, como el empleo de estructuras de datos que agrupen los elementos que forman la escena, métodos estadísticos, análisis geométrico, etc. En la figura 3.7 se muestra una clasificación (adaptada y ampliada de [15]) de las técnicas para acelerar los cálculos en trazado de rayos.

Las técnicas empleadas para **lanzar un menor número de rayos** estudian las características de la escena para decidir en qué partes es necesario el trazado de un mayor número de

rayos porque las diferencias entre píxeles vecinos son importantes (en el caso de los optimizadores estadísticos para el *antialiasing*), o bien deciden cuándo es conveniente no descender más en el trazado recursivo de rayos (en lugar de utilizar un número fijo de rebotes). Los métodos de **rayos generalizados** (como los métodos de *cone tracing*, *beam tracing* o *pencil tracing*) se basan en lanzar rayos con una descripción diferente a la basada en vector y punto de origen.

Las alternativas que más optimizan el tiempo de cómputo tratan de encontrar **intersecciones más rápidas** rayo-objeto. En esta categoría distinguimos los métodos que tratan de **acelerar el cálculo de la intersección** rayo-objeto empleando cajas límite o definiendo de forma paramétrica los objetos (por ejemplo, utilizar la descripción paramétrica de un toroide en vez de la geometría discretizada como un conjunto de triángulos). Estas técnicas son difíciles de aplicar con objetos complejos, por lo que gran parte de los esfuerzos se han centrado en tratar de calcular un **menor número de intersecciones**, que veremos con más detalle a continuación.

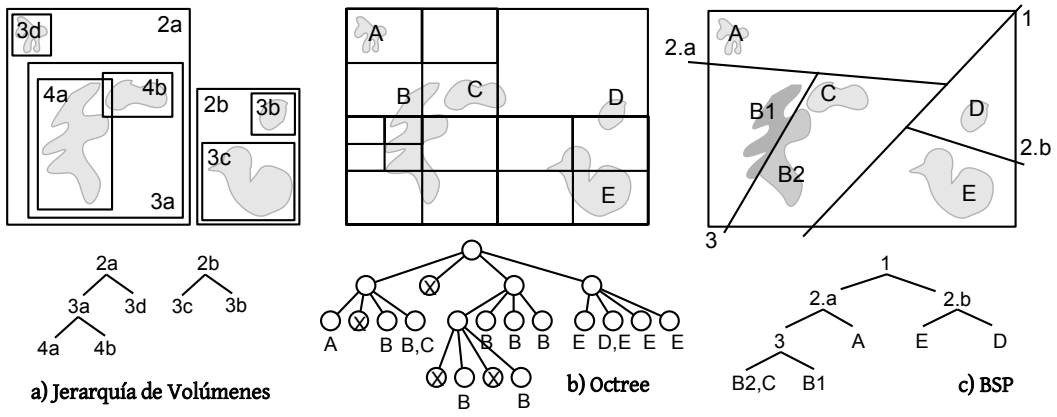


Figura 3.8: Esquemas de construcción de algunas estructuras de división espacial

Las **jerarquía de volúmenes límite** (Bounding Volume Hierarchy BVH) calculan el volumen límite (típicamente una esfera o una caja) que contiene al objeto y agrupan estos volúmenes jerárquicamente. De esta forma en la jerarquía cada nodo es un volumen límite que contiene una lista de punteros a nodos hijo (la jerarquía puede hacerse binaria o no). Cada nodo hoja de la jerarquía contiene a un objeto de la escena. Así, cuando el árbol está construido, calcu-

lamos la intersección del rayo con los nodos del primer nivel de la jerarquía. Los volúmenes de primer nivel que no intersecten con el rayo serán descartados, y continuaremos estudiando la intersección del rayo con los subnodos de aquellos nodos padre que intersectaron con él. Si la heurística elegida para la construcción jerárquica de este árbol es buena, este tipo de aproximación se comporta muy eficientemente, como en el caso del algoritmo de Goldsmith y Salmon [16]. En la figura 3.8.a) puede verse un ejemplo de este tipo de estructura.

Con una aproximación análoga a la anterior se encuentran las **aproximaciones de división espacial**. Se parte de un volumen que agrupa todos los elementos de la escena y progresivamente se divide el volumen en partes de menor tamaño, agrupando los objetos en esos volúmenes (frente a la jerarquía de volúmenes límite en la que los volúmenes se agrupan según los objetos de la escena). La partición del espacio puede realizarse en elementos (voxels) del mismo tamaño (por ejemplo, el algoritmo 3DDDA) o de forma no uniforme atendiendo a la densidad de objetos en cada región del espacio. Entre los métodos más empleados de división espacial adaptativa tenemos los árboles octales (*Octree*) (ver figura 3.8.b), los árboles de división binaria del espacio BSP (ver figura 3.8.c) o los árboles KD, muy empleados en motores de render basados en mapas de fotones [19].

Ambient Occlusion

El empleo del término de luz ambiente viene aplicándose desde el inicio de los gráficos por computador como un método muy rápido de simular la contribución de luz ambiental que proviene de todas las direcciones. La técnica de *Ambient Occlusion* es un caso particular del uso de pruebas de oclusión en entornos con iluminación local para determinar los efectos difusos de iluminación. Estas técnicas fueron introducidas inicialmente por Zhukov [33] como alternativa a las técnicas de radiosidad para aplicaciones interactivas (videojuegos), por su bajo coste computacional.

En el esquema de la figura 3.9 podemos ver en qué se basan estas técnicas. Desde cada punto P de intersección con cada superficie (obtenido mediante trazado de rayos), calculamos el valor de ocultación de ese punto que será proporcional al número de rayos que alcanzan el "*cielo*" (los que no intersectan con ningún objeto dada una distancia máxima de intersección). En el caso de la figura serán $4/7$ de los rayos lanzados. En la figura 3.9 izquierda puede

verse el esquema de cálculo del valor de ocultación $W(P)$. En 3.9 derecha (a) se muestra el resultado de un render obtenido con RayTracing e iluminación de dos fuentes puntuales, mientras que en (b) se muestra el resultado obtenido con la misma configuración que en (a) pero con Ambient Occlusion de 10 muestras por píxel.

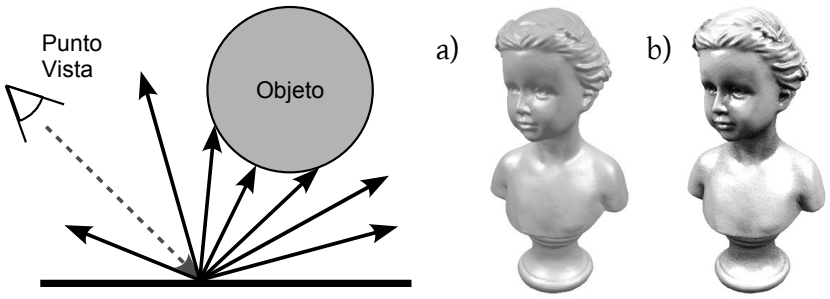


Figura 3.9: Esquema de cálculo del valor de ocultación $W(P)$ (izquierda) y comparación entre RayTracing y Ambient Occlusion (derecha).

Podemos definir la ocultación de un punto de una superficie como:

$$W(P) = \frac{1}{\pi} \int_{\omega \in \Omega} \rho(d(P,\omega)) cos\theta d\omega \tag{3.15}$$

Obteniendo un valor de ocultación $W(P)$ entre 0 y 1, siendo $d(P,\omega)$ la distancia entre P y la primera intersección con algún objeto en la dirección de ω. $\rho(d(P,\omega))$ es una función con valores entre 0 y 1 que nos indica la magnitud de iluminación ambiental que viene en la dirección de ω, y θ es el ángulo formado entre la normal en P y la dirección de ω.

Estas técnicas de ocultación (*obscurances*) se desacoplan totalmente de la fase de iluminación local, teniendo lugar en una segunda fase de iluminación secundaria difusa. Se emplea un valor de distancia para limitar la ocultación únicamente a polígonos cercanos a la zona a sombrear mediante una función. Si la función toma valor de ocultación igual a cero en aquellos puntos que no superan un umbral y un valor de uno si están por encima del umbral, la técnica de ocultación se denomina *Ambient Occlusion*. Existen multitud de funciones exponenciales que se utilizan para lograr estos efectos.

La principal ventaja de esta técnica es que es bastante más rápida que las técnicas que realizan un cálculo correcto de la iluminación indirecta. Además, debido a la sencillez de

los cálculos, pueden realizarse aproximaciones muy rápidas empleando la GPU, pudiendo utilizarse en aplicaciones interactivas. El principal inconveniente es que no es un método de iluminación global y no puede simular efectos complejos como caústicas o contribuciones de luz entre superficies con reflexión difusa.

Radiosidad

En esta técnica se calcula el intercambio de luz entre superficies. Esto se consigue subdividiendo el modelo en pequeñas unidades denominadas *parches*, que serán la base de la distribución de luz final. Inicialmente los modelos de radiosidad calculaban las interacciones de luz entre superficies difusas (aquellas que reflejan la luz igual en todas las direcciones) [18], aunque existen modelos más avanzados que tienen en cuenta modelos de reflexión más complejos.

El modelo básico de radiosidad calcula una solución independiente del punto de vista. Sin embargo, el cálculo de la solución es muy costoso en tiempo y en espacio de almacenamiento. No obstante, cuando la iluminación ha sido calculada, puede utilizarse para renderizar la escena desde diferentes ángulos, lo que hace que este tipo de soluciones se utilicen en visitas interactivas y videojuegos en primera persona actuales. En el ejemplo de la figura 3.10, en el techo de la habitación se encuentran las caras poligonales con propiedades de emisión de luz. Tras aplicar el proceso del cálculo de radiosidad obtenemos la malla de radiosidad que contiene información sobre la distribución de iluminación entre superficies difusas.

En el modelo de radiosidad, cada superficie tiene asociados dos valores: la intensidad luminosa que recibe, y la cantidad de energía que emite (energía radiante). En este algoritmo se calcula la interacción de energía desde cada superficie hacia el resto. Si tenemos n superficies, la complejidad del algoritmo será $O(n^2)$. El valor matemático que calcula la relación geométrica entre superficies se denomina **Factor de Forma**, y se define como:

$$F_{ij} = \frac{cos\theta_i cos\theta_j}{\pi r^2} H_{ij} dA_j \tag{3.16}$$

Siendo F_{ij} el factor de forma de la superficie i a la superficie j, en el numerador de la fracción definimos el ángulo que forman las normales de las superficies, πr^2 mide la distancia entre las superficies, H_{ij} es el parámetro de visibilidad, que valdrá uno si la superfice j es

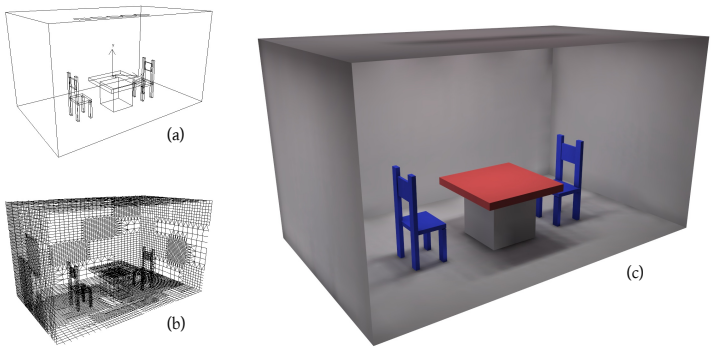


Figura 3.10: Ejemplo de uso de radiosidad. a) Malla original. b) Malla de radiosidad. c) Resultado del proceso de renderizado.

totalmente visible desde i, cero si no es visible y un valor entre uno y cero según el nivel de oclusión. Finalmente, dA_j indica el área de la superfice j (no tendremos el mismo resultado con una pequeña superficie emisora de luz sobre una superfice grande que al contrario).

Será necesario calcular n^2 factores de forma (no cumple la propiedad conmutativa) debido a que se tiene en cuenta la relación de área entre superficies. La matriz que contiene los factores de forma relacionando todas las superficies se denomina *Matriz de Radiosidad*. Cada elemento de esta matriz contiene un factor de forma para la interacción desde la superficie indexada por la columna hacia la superficie indexada por la fila (ver figura 3.11). En cada posición de la matriz se calcula el *Factor de Forma*. La diagonal principal de la matriz no es calculada.

En general, el cálculo de la radiosidad es eficiente para el cálculo de distribuciones de luz en modelos simples con materiales difusos, pero resulta muy costoso para modelos complejos (debido a que se calculan los valores de energía para cada parche del modelo) o con materiales no difusos. Además, la solución del algoritmo se muestra como una nueva malla poligonal que tiende a desenfocar los límites de las sombras. Esta malla poligonal puede ser inmanejable en complejidad si la representación original no se realiza con cuidado o si el modelo es muy grande, por lo que este tipo de técnicas no se utilizan en la práctica con modelos complejos.

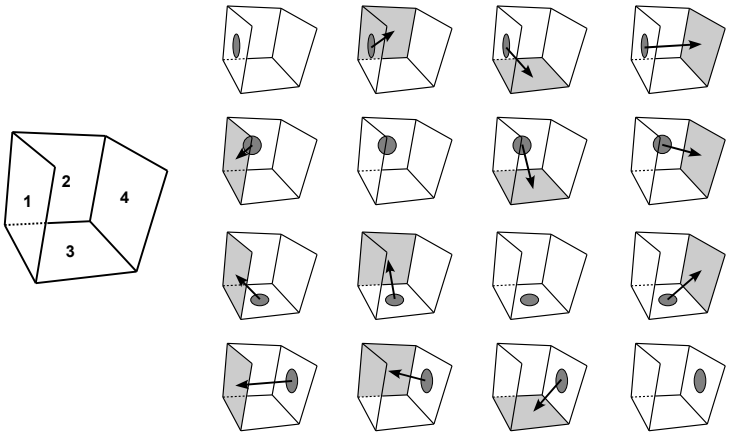


Figura 3.11: Esquema de la matriz de radiosidad

PathTracing

El método de PathTracing fue introducido por Kajiya [20], basándose en las ideas de [13]. El método *Distributed RayTracing* traza rayos distribuidos alteatoriamente para conseguir sombras suaves, *motion blur* y profundidad de campo. El método de PathTracing utiliza esta idea para calcular *todos* los posibles caminos de luz.

Algunos problemas de integración del método que no quedaban resueltos como el cálculo de la iluminación indirecta obtenida por el rebote de la luz en una superficie difusa, son resueltos por el PathTracing trazando un rayo aleatorio dentro del dominio de integración para estimar el valor de esta integral. La resolución de la función que queremos conocer (la distribución de la luz) se resuelve trazando rayos estocásticamente por todos los caminos de luz posibles[3] y dividiendo por el número de muestras que tomamos por cada píxel obtenemos una estimación del valor de la integral teniendo en cuenta *todos* los posibles caminos de luz en ese píxel. Este método de resolución se basa en la técnica de integración de Monte Carlo.

El método de integración de Monte Carlo emplea muestras aleatorias de la función a evaluar $f(x)$ para estudiar sus propiedades. Dada la siguiente integral:

$$I = \int_a^b f(x)dx \tag{3.17}$$

una forma intuitiva de resolverla sería calculando la media de $f(x)$ en el intervalo de a a

[3] Debido a que la solución exacta por otros métodos de integración no es abordable.

b y multiplicar esta media por la longitud del intervalo $(b-a)$. Esto nos daría una primera aproximación bastante pobre. Sin embargo, si hacemos la media de los valores de $f(x)$ en N posiciones $\xi_1, \xi_2, ..., \xi_N$ uniformemente distribuidas entre a y b tendremos:

$$I_m = (b-a)\frac{1}{N}\sum_{i=1}^{N} f(\xi_i) \tag{3.18}$$

siendo I_m la estimación de Montecarlo para la integral I.

El algoritmo del método de PathTracing es básicamente igual al de RayTracing visto anteriormente, pero incluyendo múltiples muestras aleatorias por pixel y posiciones aleatorias en la lente de la cámara. Además, a diferencia del método de RayTracing clásico en el que las fuentes de luz eran puntuales, en el método de PathTracing se requieren luces con un cierto área y se muestrean posiciones aleatorias dentro de ese área.

Una de las principales diferencias entre este método y el RayTracing original es la construcción del camino de la luz. En PathTracing sólo se lanza un único rayo en la llamada recursiva, lo cual nos produce un camino (*Path*), y no un árbol de rayos. Este rayo puede ser un rayo difuso, un rayo especular (espejo) o un rayo transmitido, con una cierta probabilidad. Cada material tiene un nivel de reflexión difusa k_d, especular (espejo) k_s y de transmitividad k_t, que puede estar definido entre cero y uno; cero implica que no existe ese tipo de interacción con la luz. Definimos $k_m = k_d + k_s + k_t$. Para elegir qué tipo de rayo vamos a lanzar bastará con obtener un número aleatorio W entre $(0...k_m)$. Si $(W < k_d)$ trazaremos un rayo difuso; en caso contrario veremos si $(W < k_d + k_s)$, trazaremos un rayo especular; en cualquier otro caso, lanzaremos un rayo transmitido.

El principal problema del método de *PathTracing* es la lenta convergencia a la solución final, lo que se traduce en ruido. Si se conoce la distribución de las regiones más brillantes de la escena, se pueden concentrar las muestras en esas zonas para reducir la varianza rápidamente.

PathTracing Bidireccional

Fue introducido por Lafortune y Willems en 1993 [21]. Se planteó como una extensión del PathTracing (en realidad el PathTracing puede considerarse como un caso particular de esta técnica); al igual que en PathTracing, el objetivo es muestrean los posibles caminos de luz en la escena. En PathTracing además se calculan los caminos desde las fuentes de luz. Esto

aprovecha la idea de que hay ciertos caminos que son más fáciles de trazar desde las fuentes de luz, como por ejemplo el caso de las caústicas[4]. De esta forma se tienen dos subcaminos; los que comienzan en la fuente de luz, y los que comienzan en el punto de vista del observador.

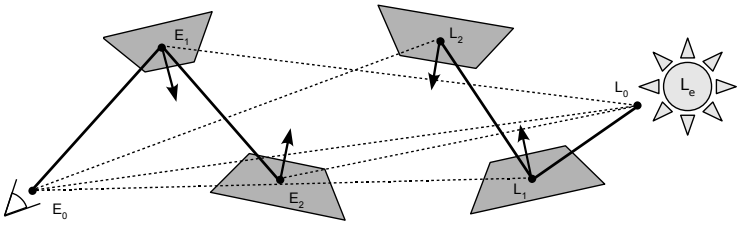


Figura 3.12: Esquema de PathTracing Bidireccional

La forma de generar los caminos es la siguiente: primero se elige un punto y una dirección aleatorios de una fuente de luz de la escena. De forma análoga al PathTracing se calculan los caminos de luz desde el observador. El camino bidireccional se calcula conectando cada vértice del subcamino del observador con cada vértice del subcamino de luz (ver figura 3.12; las líneas punteadas representan estos rayos de sombra que conectan los subcaminos). Un problema que se plantea con esta aproximación es cuando el camino desde el observador tiene longitud cero, conectando directamente el observador con el subcamino de luz, no se sabe a qué píxel va a contribuir ese camino.

Como puede verse en la figura 3.12, los rayos son trazados desde el punto de vista (E_i) y desde las fuentes de luz (L_i), y conectados mediante rayos de sombra (las líneas discontínuas).

Finalmente habrá que emplear una función que mida la contribución de los dos caminos (el camino de luz propaga *flux* y el camino desde el observador *radiancia*, siendo necesario un método de combinación).

Transporte de Luz de Metrópolis

El método de transporte de luz de Metrópolis (MLT) es un algoritmo de tipo Monte Carlo empleado para la resolución de la ecuación de renderizado aportando una solución de ilu-

[4]La probabilidad de que un rayo reflejado en una superficie atraviese un objeto transparente y llegue a la fuente de luz es muy pequeña comparada con el camino inverso (desde la fuente de luz a la superficie).

minación global. Fue propuesto en 1997 por Veach y Guibas [30]. El algoritmo consta de dos fases; en la primera se utiliza un trazado de rayos bidireccional para generar un conjunto de caminos iniciales de la luz. En una segunda fase, cada uno de estos caminos sufrirá una mutación que será aceptada o rechazada según una probabilidad.

Como hemos comentado al inicio de la sección, para generar una imagen realista por computador tenemos que resolver un problema de transporte de luz. La aproximación empleada por los algoritmos de PathTracing convergen a la solución correcta aunque sufren ruido. Hay ciertas situaciones en las que los algoritmos de PathTracing son ineficientes, como situaciones con fuertes focos de iluminación indirecta, zonas pequeñas por donde tiene que introducirse la luz, etc. Por contra, estas situaciones son manejadas correctamente por el método de transporte de luz de Metrópolis (MLT).

La idea básica es mutar los caminos que llegan a la imagen para conseguir nuevos caminos exitosos. Esta "mutación" consiste en eliminar vértices de un camino o añadir nuevos. Una vez que se ha encontrado un camino exitoso, se exploran caminos cercanos a él. Cada mutación será aceptada o rechazada según una probabilidad, dependiendo de su contribución a la imagen final. Existen diferentes estrategias sobre cómo mutar un camino. La elección de una u otra dependerá de las características de la escena a renderizar.

El camino mutado es aceptado o no según una función que tiene en cuenta el camino actual y el camino mutado. Si el camino es aceptado, el nuevo camino reemplaza al antiguo; si es rechazado, simplemente se desecha. Las mutaciones pueden ser casi arbitrarias porque las probabilidades de aceptación aseguran que los caminos mutados se distribuyen de forma proporcional a su contribución a la imagen final. Todos los caminos posibles de la escena deben tener una probabilidad de ser explorados mayor que cero (para evitar que algunos caminos puedan ser ignorados).

Para la generación de imágenes sin ruido empleando el algoritmo MLT debemos disponer de buenas funciones de generación de mutaciones. Las mutaciones deben tener un tamaño adecuado, no demasiado grandes ya que la probabilidad de aceptación de mutaciones muy grandes es baja. Según Veach [30], las estrategias de mutación elegidas deben cumplir las siguientes propiedades:

- **Alta probabilidad de aceptación**: Si se rechazan muchas mutaciones, se aumenta la

varianza de la solución. Además se desperdicia tiempo de cómputo mutando siempre el mismo camino.

- **Evitar mínimos locales**: Para obtener resultados correctos, es importante explorar todos los posibles caminos con probabilidad mayor que cero; de otra forma podríamos caer en mínimos locales en regiones del espacio y no explorar otras regiones que en realidad podrían ser muy importantes.
- **Cambios en la localización**: Las mutaciones deben cambiar la posición en el plano de imagen para que todos los píxeles de la misma sean evaluados. Además hay que tener en cuenta que la estrategia de *sampling* sea buena para evitar el *aliasing* (las muestras deben estar bien distribuidas).
- **Bajo coste**: La estrategia de mutación debería utilizar el menor número de operaciones de trazado de rayos posibles, ya que éstas requieren un alto tiempo de cómputo.

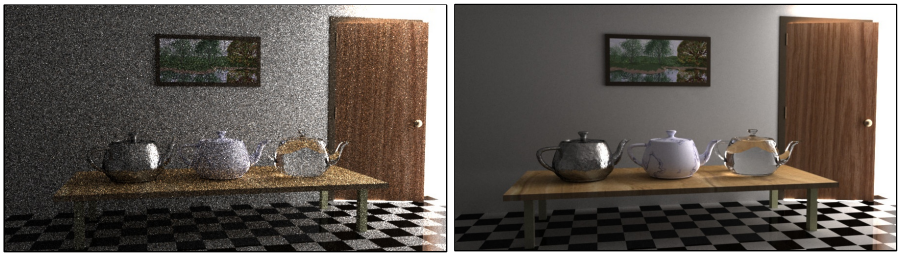


Figura 3.13: **Izquierda:** Imagen obtenida con un Path Tracer Bidireccional (40 muestras por píxel). **Derecha:** MLT (250 mutaciones por píxel). Ambas imágenes emplearon el mismo tiempo de cómputo. Ejemplo extraído de [30].

En general podemos afirmar que el método MLT, con el mismo tiempo de cómputo obtiene mejores resultados que un trazador de rayos bidireccional (ver figura 3.13), debido principalmente a que MLT reutiliza partes de caminos que contribuyen fuertemente a la escena, explotando así las propiedades de coherencia espacial. El algoritmo es robusto y eficiente para situaciones de iluminación difíciles, como escenas con una iluminación indirecta muy fuerte o caústicas. Resulta sencillo ampliar el algoritmo básico y añadir nuestras propias estrategias de mutación.

Photon Mapping

Este método fue propuesto por Jensen [19] y se basa en desacoplar la representación de la iluminación de la geometría. Se realiza en dos pasos, primero se construye la estructura del mapa de fotones (trazado de fotones), desde las fuentes de luz al modelo (ver figura 3.14 izquierda). En una segunda estapa de render se utiliza la información del mapa de fotones para realizar el renderizado de manera más eficiente (ver figura 3.14 derecha).

Cuando se emite un fotón, éste es trazado a través de la escena de igual forma que se lanzan los rayos en raytracing, excepto por el hecho de que los fotones propagan *flux* en vez de *radiancia*. Cuando un fotón choca con un objeto puede ser reflejado, transmitido o absorbido (según las propiedades del material y un factor aleatorio dentro del dominio del material).

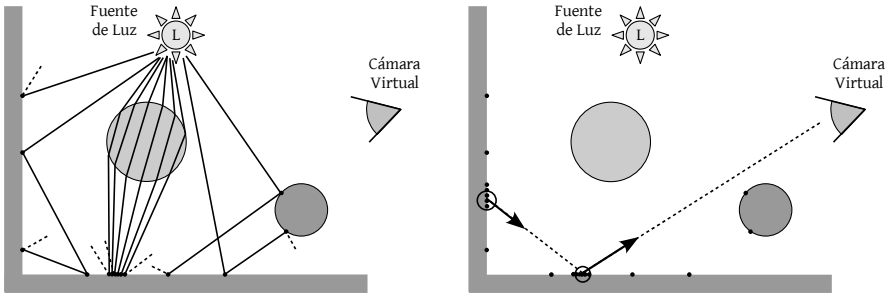


Figura 3.14: Las dos etapas en el algoritmo básico del mapeado de fotones: construcción del mapa de fotones (izquierda) y utilización de su información para renderizar la escena (derecha).

Veamos a continuación las dos etapas en detalle. En la primera pasada se realiza el cálculo del mapa de fotones. En realidad, este mapa se divide en dos estructuras; una para guardar los impactos de los fotones debido a las caústicas (*causitics photon map*) y otro para representar la iluminación global (*global photon map*). El **mapa de fotones de caústicas** contiene los fotones que han sido reflejados o transmitidos al chocar con una superficie especular antes de chocar con una difusa. En la figura 3.14, los fotones que golpean a la esfera de la izquierda (con propiedades de transmitividad) la atravesarían y al impactar después con el suelo formarían el mapa de fotones de caústicas. Este mapa de fotones se utilizará para simular

la concentración de luz debido a este fenómeno en las zonas que sean directamente visibles desde la cámara (ver figura 3.14 derecha).

El **mapa de fotones global** contiene todos los fotones que han golpeado a superficies difusas en la escena (por tanto, contiene la iluminación directa, indirecta y las caústicas). Como el mapa de fotones de caústicas está *contenido* en el mapa de fotones global, hay que tener cuiado en no tener en cuenta dos veces estos fotones. Los fotones son reflejados por las superficies difusas de la escena, pero no por las especulares (en la figura 3.14 izquierda, queda reflejado este fenómeno ya que ningún fotón que toca la esfera transparente rebota; simplemente la atraviesa).

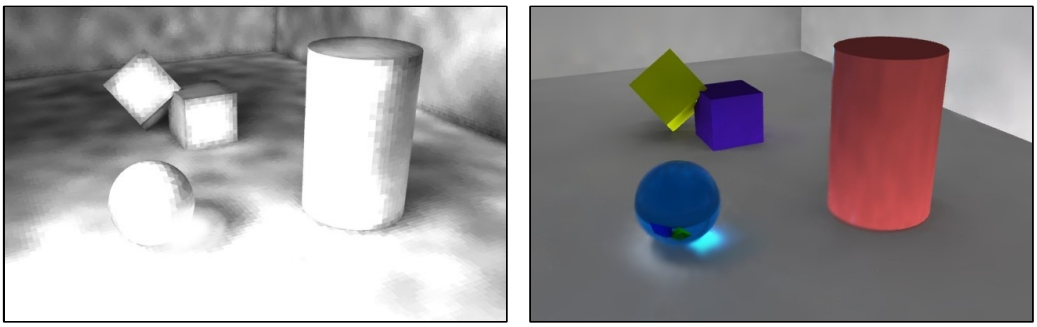


Figura 3.15: Ejemplo de uso de Mapeado de Fotones. **Izquierda**: Distribución de los impactos de los fotones en el mapa de fotones global. **Derecha**: Resultado final; se aprecian las caústicas bien calculdas según la dirección de la luz en la esfera transparente de la izquierda.

En la segunda etapa se renderiza empleando RayTracing distribuido (o PathTracing), con varias estimaciones por píxel. Primero se calcula de forma precisa la iluminación directa, de forma similar a como se realizó en RayTracing. De igual forma se tiene en cuenta la reflexión especular (o *glossy*) empleando el cálculo preciso del vector de reflexión que se utilizaba en RayTracing distribuido. Las posibles caústicas de la escena se calculan empleando el mapa de fotones de caústicas que se calculó en la primera etapa. No se utiliza nunca el método de PathTracing para calcular las caústicas debido a que es muy ineficiente (ver figura 3.15, se percibe el impacto de los fotones de caústicas sobre el suelo). Finalmente, la iluminación indirecta se calcula empleando el mapa de fotones global. Gracias a esta información, el método de Monte Carlo converge antes a la solución correcta y se suaviza enormemente el ruido que presentan

este tipo de métodos. Mediante el método del mapa de fotones, la iluminación global se calcula mediante estos cuatro componentes: $I_{Global} = I_{Directa} + R_{Especular} + Causticas + I_{Indirecta}$. En los últimos dos componentes de la ecuación anterior nos ayudamos de los mapas de fotones.

Como hemos visto anteriormente, los fotones son almacenados cuando impactan sobre superficies no especulares (para reflejos se utiliza RayTracing clásico). Este almacenamiento se realiza sobre una estructura de datos que está desacoplada de la geometría del modelo, y que se utiliza en la etapa de render. Para que la búsqueda sea rápida se suelen emplear *kd-trees* [10] (debido a que la distribución de fotones no es uniforme), cuyo tiempo medio de búsqueda con n fotones es de $O(logn)$, y $O(n)$ en el peor de los casos.

3.2.4. Consideraciones Finales

Figura 3.16: Resultados de la misma escena con diferentes métodos de render: a) RayCasting b) RayTracing (Whitted) c) Ambient Occlusion d) PathTracing (Skydome) e) PathTracing (Un foco de iluminación directa) f) PathTracing + Photon Mapping (+ Iluminación basada en imagen HDR) (ver versión a color en C.1).

En la tabla 3.2 se muestran los tiempos de cálculo de la escena de la figura 3.16. La resolución de todos los ejemplos es de 640x480 píxeles y oversampling de 8 muestras por píxel. La

escena está formada por 51016 polígonos. El modelo del dragón tiene propiedad de reflexión (2 niveles de recursión), y las copas de reflexión (3 niveles) y refracción (4 niveles). La imagen ha sido renderizada en un Centrino 2GHz, con 1 GB de RAM bajo Debian GNU/Linux. El motor de render utilizado para los métodos de iluminación global (PathTracing y Photon Mapping) es Yafray 0.0.8, en los otros métodos se utilizó el motor interno de Blender 2.41.

Método	Tiempo *(min:seg)*
RayCasting	00:04
RayTracing	Octree (*64 nodos*): 01:34
	Octree (*128 nodos*): 01:16
	Octree (*256 nodos*): 01:14
	Octree (*512 nodos*): 01:23
Ambient Occlusion	Octree (*64 nodos*): 14:02
	Octree (*128 nodos*): 10:15
	Octree (*256 nodos*): 11:22
	Octree (*512 nodos*): 16:17
PathTracing (SkyDome)	*16 Muestras*: 09:53
	24 Muestras: 10:23
	36 Muestras: 16:28
	64 Muestras: 27:37
	128 Muestras: 39:03
PathTracing + Photon Mapping	*128 Muestras*: 29:41
	256 Muestras: 40:30
	512 Muestras: 53:12
	1024 Muestras: 94:58
	2048 Muestras: 170:17

Cuadro 3.2: Tiempos de render según métodos y parámetros de calidad.

Como puede verse en la tabla 3.2, los tiempos de generación de la imagen son muy dependientes de la técnica de render empleada. Así, por ejemplo hay una diferencia de más de

dos órdenes de magnitud entre una imagen generada con *PathTracing* empleando 2048 muestras por píxel y el método de *RayTracing clásico* para la misma escena. Incluso dentro de una misma técnica de generación, el tiempo invertido para su obtención es sustancialmente diferente (por ejemplo, el tiempo que se emplea es proporcional al número de muestras en PathTracing, pero no lo es en la calidad percibida por el sistema de visión humano).

Emplearemos estas diferencias notables en los tiempos de render guiados por conocimiento experto para modelar en nuestra arquitectura un sistema multiagente que decida qué técnica y con qué calidad se realizará la generación de cada región de una escena tridimensional.

Capítulo 4

Metodología de Trabajo

4.1. Introducción

Para poder proporcionar estimaciones del coste y el rendimiento del proceso de render, así como sugerencias en términos de configuraciones del proceso, es necesario realizar un análisis de qué características de un modelo 3D y parámetros de render afectan en mayor grado al tiempo de render y la calidad del resultado producido. Estas características pueden

ser clasificadas en dos grandes grupos: **características del modelo** y **características del proceso de render**.

Las características del modelo son aquellos atributos relativos a la definición del modelo 3D en sí, como número de objetos o polígonos, número de fuentes de iluminación y su localización o texturas y materiales empleados. Aunque es cierto que estos aspectos tienen un gran impacto en el coste y rendimiento del proceso, alterar estos atributos daría lugar a un cambio en el modelo creado por el usuario, por lo que no forma parte de los objetivos del presente proyecto alterar dichas propiedades.

Por otro lado, las características del proceso de render son todos aquellos atributos que definen el comportamiento exacto del algoritmo empleado en el render del modelo. Ejemplos de estas características son el número de muestras empleadas en la simulación de la luz, profundidad del algoritmo recursivo de trazado de rayos, etc. Estos son los atributos interesantes para el análisis que tiene lugar en este proyecto, ya que no alteran de ningún modo la naturaleza del modelo creado por el usuario y una buena configuración de estos es crucial a la hora de producir resultados de alta calidad en un tiempo *razonable*.

Un modelo 3D puede ser literalmente *cualquier cosa*, desde un objeto, a un edificio o un paraje natural. Debido esta variedad, se hace necesario acotar el tipo de modelo que se va a tratar en este estudio. Una de las utilizaciones de los gráficos por computador que más demandadas están hoy en día es la creación de modelos virtuales de viviendas en construcción. Con esta herramienta, los constructores pretenden mostrar a sus clientes cómo serán sus viviendas cuando su construcción esté finalizada mediante visitas virtuales, imágenes o vídeos. El render de este tipo de modelos es especialmente complejo cuando se trata de procesar escenas de interior, es decir, modelos en los que la luz entra a través de unos pocos huecos en objetos opacos. En este tipo de modelos se hace forzosamente necesaria la utilización de algoritmos de iluminación global, es decir, aquellos que toman en cuenta tanto los impactos de la luz provenientes directamente de las fuentes de iluminación, como aquellos que provienen de los rebotes de la luz en otros objetos. Este tipo de iluminación se denomina **iluminación indirecta**. Es en este tipo de algoritmos donde una buena configuración de los parámetros de entrada se hace crucial, ya que tiene un gran impacto en el tiempo de ejecución del proceso y en la calidad de los resultados producidos.

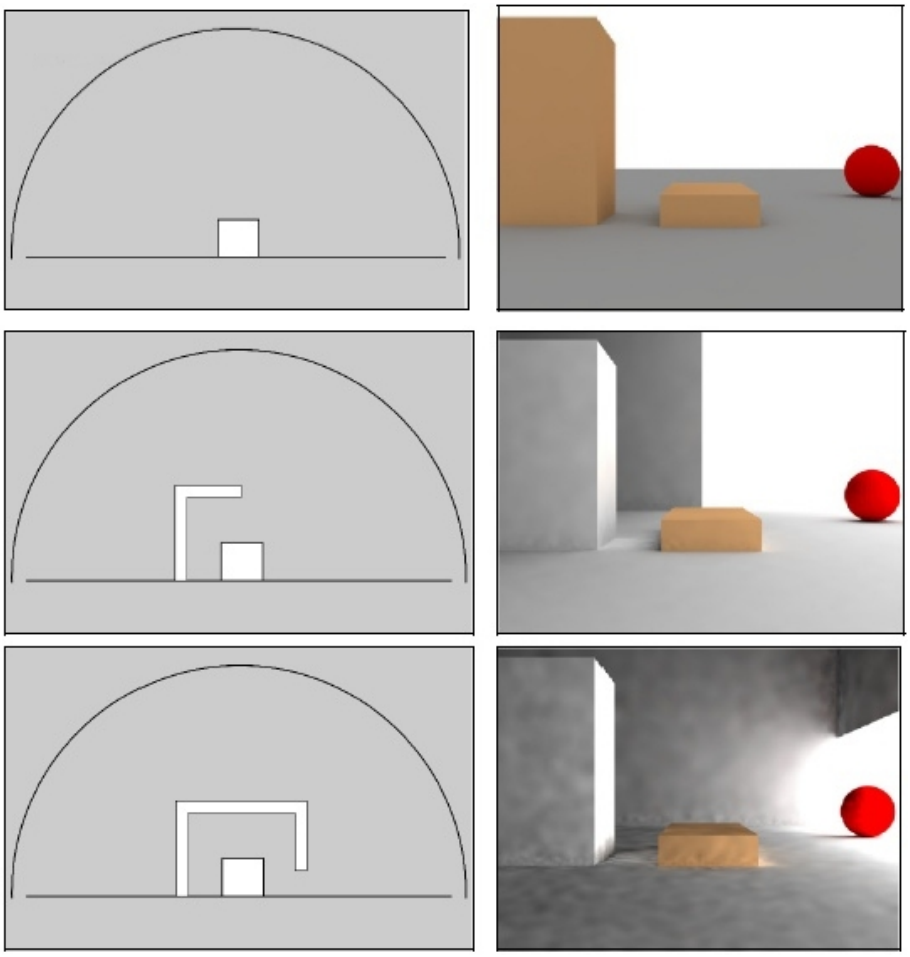


Figura 4.1: Efecto en los resultados de diferentes configuraciones de escena

Así, los modelos estudiados en este proyecto serán escenas de interior donde las fuentes de luz estén situadas tanto dentro, como fuera de la escena en sí. La problemática de estos modelos deriva del funcionamiento de los algoritmos de iluminación global. Por razones de eficiencia, en estos se simula el comportamiento físico de la luz en sentido opuesto. Es decir, mientras que en realidad los rayos de luz son trazados desde las fuentes de iluminación, impactan en los objetos de nuestro entorno y llegan a nuestros ojos, en la simulación realizada por este tipo de algoritmos los rayos de luz son emitidos por la cámara virtual (o punto de vista), impactan en los objetos definidos en el modelo y llegan (algunos) hasta las fuentes de iluminación. Son estos rayos que alcanzan las fuentes de iluminación los que el algoritmo tiene en cuenta a la hora de establecer los colores de la escena que darán lugar a la imagen

generada. Por este motivo, las escenas de interior son especialmente sensibles a *ruido*, lo que se traduce en resultados de baja calidad. En la figura 4.1 se pueden observar diferentes configuraciones de escena que dificultan en menor o mayor grado el impacto de los rayos de luz con las fuentes de iluminación (columna izquierda) y los respectivos resultados producidos (columna derecha).

El proceso de desarrollo del presente proyecto está dividido en dos etapas bien diferenciadas:

- **Generación de la base de conocimiento**: El objetivo de esta etapa es obtener un conjunto de datos para entrenar los distintos métodos de aprendizaje máquina utilizados. En esta etapa se definen los atributos por los que cada ejemplar del conjunto de datos está compuesto, se generan grandes cantidades de ejemplares de render y se crean los conjuntos de datos de entrenamiento.

- **Desarrollo de YAReW**: Esta etapa tiene como objetivo desarrollar un sistema que responda a las necesidades identificadas en capítulo 2. En la sección 4.3 se describe la metodología de desarrollo utilizada.

La tecnología utilizada en el desarrollo del sistema ha sido:

- **Blender v2.45**: Blender es un software de desarrollo integral de gráficos por computador. Incluye facilidades para el modelado de geometrías, la descripción de escena y el renderizado. [4]

- **Yafray v0.0.9**: Yafray es un motor de render de iluminación global, basado en los métodos de Path Tracing y Photon Mapping (ver 3.2.3). [6]

- **Orange v0.9.66**: Orange es un software de análisis y procesamiento de datos mediante técnicas de Minería de Datos y Aprendizaje Máquina, con una arquitectura de componentes y desarrollado en la Universidad de Ljubljana, Eslovenia. Incluye varias técnicas de preproceso, modelado y exploración de datos. Así mismo, implementa una gran variedad de métodos de aprendizaje máquina, como árboles de decisión, clasificadores bayesianos y métodos basados en clasificación por vecindad. Una de las mayores ventajas de Orange frente a otros paquetes de software del mismo tipo es que puede utilizarse

mediante una interfaz de programación visual (Orange Canvas) y mediante una interfaz de programación de aplicaciones (API) escrita en lenguaje Python, lo que permite el desarrollo de aplicaciones que hagan uso de las funcionalidades proporcionadas por este software. Por otro lado, el núcleo del sistema está escrito en C++, lo que lo hace muy eficiente cuando se trata de analizar grandes cantidades de datos. Orange se distribuye bajo los términos de la licencia GPL v2.0. Todo esto lo hace especialmente adecuado para su utilización en este proyecto. [2]

- **Python v2.44**: Python es un lenguaje de programación orientado a objetos e interpretado. Su eficiencia, la completitud de su biblioteca de funciones y la facilidad para ser ejecutado sobre distintas plataformas e interaccionar con componentes software escritos en otros lenguajes le ha convertido en una alternativa muy atractiva a los lenguajes de programación clásicos en varios tipos de desarrollo. [7]

Todo el software utilizado en el desarrollo del presente proyecto está publicado bajo licencias de código abierto, reconocidas por la *Open Source Initiative* [1].

4.2. Generación de la base de conocimiento

Los métodos de clasificación y estimación basados en aprendizaje automático necesitan grandes cantidades de datos para inducir hipótesis que describan el modelo subyacente lo más precisamente posible. Uno de los mayores problemas en el desarrollo de YAReW deriva de este hecho, y ha sido conseguir una base de conocimiento con un número de datos relativamente grande y completa.

El objetivo de esta etapa es obtener dicha base del conocimiento. Cada dato en la base de conocimiento es una instancia del proceso de render sobre una escena determinada y caracterizada por una colección de atributos. Esto plantea ciertos problemas, ya que dos instancias de render lanzadas con los mismos valores de parámetros de entrada sobre escenas muy diferentes tendrán resultados diferentes en términos de tiempo de render y calidad de resultados.

Por este motivo, la base de conocimiento está formada por un conjunto de escenas de

diferentes características. Cada escena dispone de un conjunto de instancias, que son los conjuntos de datos utilizados por los métodos de aprendizaje máquina implementados en el sistema. De esta forma se pretende solucionar el problema descrito anteriormente. Cuando un usuario proporciona una nueva escena al sistema, este tratará de identificar sus características generales en términos de geometría, grado de iluminación proveniente de fuentes internas y externas, etc. Una vez identificadas dichas características, el sistema tratará de encontrar la escena contenida en su base de datos que sea más similar a la escena de usuario en una etapa llamada de *matching*. Así, el sistema utilizará el conjunto de datos de la escena más afín a la proporcionada por el usuario. De esta forma se obtiene una buena aproximación en la estimación del tiempo de render y la calidad del resultado, ya que dos instancias del proceso de render con los mismos parámetros de entrada lanzadas sobre dos escenas similares deben proporcionar unos resultados de aproximadamente la misma calidad empleando aproximadamente el mismo tiempo de render. En la figura 4.2 puede verse la estructura de la base de conocimiento.

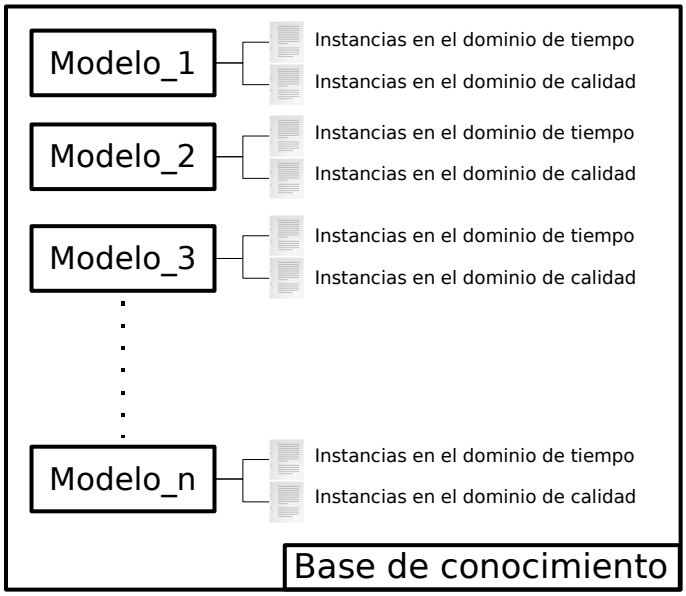
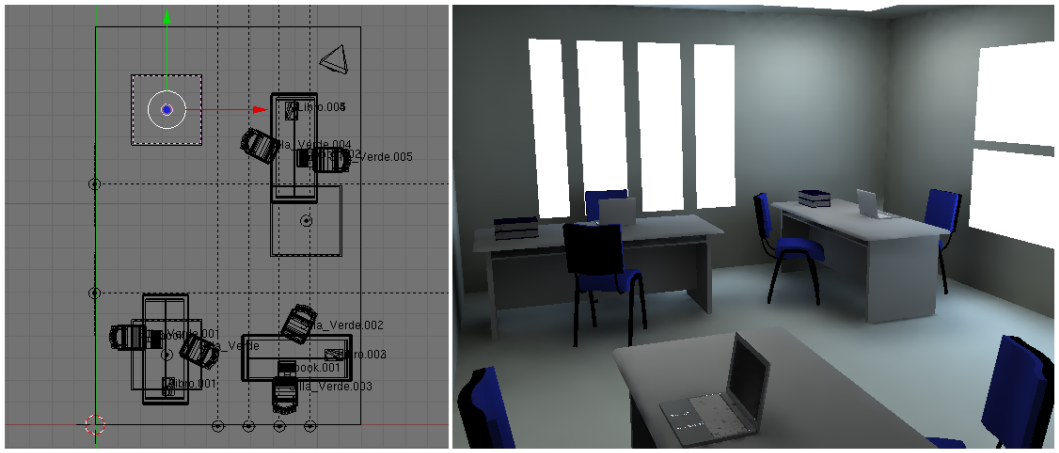


Figura 4.2: Estructura de la base de conocimiento manejada por YAReW

Obtener un conjunto numeroso de escenas diferentes no es una tarea trivial. Las posibles

alternativas para solucionar este problema son: solicitar la ayuda a la comunidad para que los usuarios de *Yafray*, aporten sus escenas al proyecto, o bien implementar una aplicación que genere un conjunto de escenas automáticamente. Si bien la primera alternativa podría dar como resultado un conjunto de escenas más significativo desde el punto de vista de un experimento real, tiene como inconveniente la dificultad de conseguir un buen número de escenas en un plazo relativamente corto (además, intervienen aspectos psicológicos, ya que los usuarios suelen ser reticentes a proporcionar los ficheros fuentes de su trabajo). Por otro lado, el desarrollo de una herramienta de generación automática de escenas es un proceso no trivial y relativamente limitado, ya que es muy difícil generar de forma automática escenas con grandes diferencias en cuanto a geometría.



(a) Modelo 3D de una escena generada automáticamente

(b) Resultado obtenido al renderizar el modelo de la izquierda

Figura 4.3: Modelo generado automáticamente con el generador de escenas automático desarrollado

Aún con las limitaciones mencionadas, se ha optado por desarrollar una herramienta de generación automática de escenas de interiores, debido a las ventajas de poder disponer del número de escenas que se deseen y de realizar nuestro experimento en un entorno más controlado. En la figura 4.3 puede verse un ejemplo de escena generada con la herramienta desarrollada.

En la figura 4.4 puede verse la colección de etapas que forman una iteración del proceso

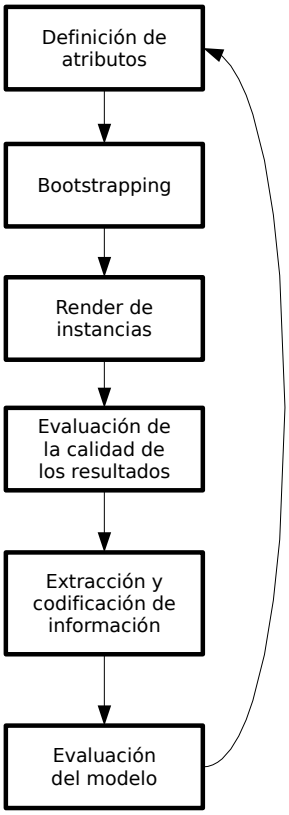


Figura 4.4: Proceso de generación de la base de conocimiento

de generación de la base del conocimiento. A continuación se describe cada una de estas etapas.

4.2.1. Definición de atributos

Cada instancia del proceso de render está caracterizada por una colección de atributos que describen las particularidades de la instancia en sí. Los atributos más interesantes para incluir en esta colección son aquellas características que afectan en mayor medida tanto al tiempo de ejecución del proceso de render, como a la calidad de la imagen obtenida.

Son muchas las características de un modelo 3D que afectan al coste y rendimiento del proceso de render en gran medida. Podemos clasificar todas ellas en dos grandes grupos: *características de la escena* y *características del proceso de render*.

Las **características de la escena** son aquellas relativas a la definición del modelo 3D, como número de objetos y polígonos, localización de estos, número de fuentes de iluminación y localización de estas, materiales, texturas y *shaders*[1] utilizados. Estas características tienen un gran impacto en el coste y el rendimiento del proceso de render, debido a la simulación del comportamiento físico de la luz que realizan los algoritmos de iluminación global. Además, estas características son definidas por el usuario a la hora de crear el modelo 3D y son precisamente las que lo definen.

Por otro lado, las **características del proceso de render** son todos aquellos parámetros de entrada del algoritmo de render utilizado. Estas características no definen el modelo 3D en sí, sino que definen el comportamiento específico del algoritmo utilizado para renderizar la escena. Algunos ejemplos de los parámetros más comúnmente utilizados son número de muestras por píxel, profundidad de recursividad en iluminación directa e indirecta o número de rayos o fotones a lanzar por fuente de iluminación.

Así, la diferencia clave entre las características específicas de escena y las del proceso de render es que mientras que las primeras definen la naturaleza del modelo en sí, las segundas tan sólo afectan al comportamiento exacto del proceso de render.

El objetivo del presente proyecto es proporcionar soporte a la decisión al usuario en forma de configuraciones de parámetros de render que maximicen la relación $\frac{Rendimiento}{Coste}$, sin modificar el modelo 3D definido por el usuario. Por este motivo, los atributos descriptores de cada instancia del proceso de render en YAReW serán aquellas características del proceso de render que mayor impacto tengan en el coste y el rendimiento de dicho proceso. Está fuera del ámbito de este proyecto estudiar cómo cambios en la geometría descriptiva de un modelo 3D afectan al resultado final del render.

Parámetro	Posibles valores	Significado
OSA	5, 8, 11, 16	Establece el nivel de *oversampling*. Suaviza las transiciones entre píxeles de diferente color.

Continúa en la siguiente página...

[1] La palabra *shader* hace referencia a los algoritmos de sombreado utilizados en el proceso de render. Se ha optado por utilizar el término anglosajón por ser este extensivamente utilizado en la literatura en castellano.

Parámetro	Posibles valores	Significado
Method	Skydome	No contempla los rebotes de luz entre objetos. Es un método muy apto para escenas de exteriores. Muy rápido, aunque consigue resultados no tan realistas.
	Full	Tiene en cuenta los rebotes de luz entre objetos. Este método es el más apropiado para escenas de interior. Es muy realista, pero también muy costoso.
GIQuality	None, Low, Medium, High, Higher, Best,	Establece el número de muestras tomadas por el trazador de rayos.
Depth	[1,8]	Número de rebotes de la luz (profundidad de recursividad) para rayos de iluminación indirecta (rebotes de luz entre objetos).
CDepth	[1,8]	Número de rebotes de la luz (profundidad de recursividad) para cáusticas.
PhotonCount	[0,10000000]	Número de fotones lanzados para ayuda en la iluminación.
PhotonRadius	[0.00001,100.0]	Distancia dentro de la cual los fotones calculados tienen un valor máximo de precisión.
PhotonMixCount	[0,1000]	Número de fotones a ser tomados en cuenta dentro del radio definido.
ShadowQuality	[0.01,1.0]	Establece el valor de suavizado en sombras. Cuanto mayor es este valor, se toman más muestras y por lo tanto la calidad y el tiempo aumentan.

Continúa en la siguiente página...

Parámetro	Posibles valores	Significado
Precision	[1,50]	Precisión a nivel de píxel. El algoritmo toma una muestra irradiada cada X píxeles, siendo X el valor de este parámetro.
Refinement	[0.001,1.0]	Valor umbral de iluminación para suavizado de sombras. Cuanto menor sea este valor, mayor calidad tendrán las sombras.
EmitPower	[0.01,100.0]	Valor de escalado para rayos de iluminación directa.
GIPower	[0.01,100.0]	Valor de escalado para rayos de iluminación indirecta.
Raydepth	[1,80]	Número de rebotes de la luz (profundidad de recursividad) para rayos de iluminación directa.

Cuadro 4.1: Lista de parámetros de entrada al motor de render *Yafray*

La tabla 4.1 muestra el conjunto de parámetros de render definido por *Yafray*. A excepción del parámetro *Method*, todos los demás serán los atributos descriptores de cada una de las instancias de render manejadas por la base de conocimiento del sistema. La característica *Method* no es incluida porque cuando esta toma el valor *Skydome*, el algoritmo no calcula los rebotes de la luz entre los distintos objetos del modelo (iluminación indirecta). Esto, aunque aceptable (e incluso aconsejable) en escenas de exterior, normalmente produce resultados de calidad muy pobre en escenas de interior, donde la iluminación realista de la escena se debe en gran parte a la iluminación indirecta. Ya que el interés de este proyecto se basa en proporcionar soporte a la decisión en escenas de interiores (que son las que suponen un mayor esfuerzo de configuración), todas las instancias de render tendrán establecido el valor *Full* para la característica *Method*.

Una vez especificados qué atributos serán los que describan cada instancia de render, queda por definir qué características se desean predecir o estimar. Ya que son objetivos del

sistema proporcionar estimaciones del tiempo de render y la calidad de los resultados, además de sugerir configuraciones que maximicen la relación $\frac{\text{Calidad de Resultado}}{\text{Tiempo de Render}}$, parece natural que los atributos objetivos sean precisamente estos: **tiempo de ejecución del proceso de render** y **calidad del resultado** obtenido. El hecho de que sean dos las características a aproximar nos fuerza a definir dos dominios distintos: uno cuyo atributo objetivo será el tiempo de render y el otro la calidad del resultado.

OSA	GIQuality	Depth	CDepth	PhCount	PhRadius	PhMixCount	ShQuality	Prec	Refine	EPower	GIPower	Raydepth	Time
16	HIGHER	5	1	250000	2.0	100	0.7	10	0.85	3.0	2.5	6	289

Figura 4.5: Instancia de render proyectada sobre el dominio de tiempo

OSA	GIQuality	Depth	CDepth	PhCount	PhRadius	PhMixCount	ShQuality	Prec	Refine	EPower	GIPower	Raydepth	Quality
16	HIGHER	5	1	250000	2.0	100	0.7	10	0.85	3.0	2.5	6	7

Figura 4.6: Instancia de render proyectada sobre el dominio de calidad

En las figuras 4.5 y 4.6 puede verse un mismo ejemplar de render proyectado sobre el dominio de tiempo y calidad respectivamente.

4.2.2. Bootstrapping

El objetivo de la etapa de *bootstraping*[2] se basa en conseguir el mayor número de configuraciones de parámetros de render posible para poder obtener posteriormente un conjunto de instancias del proceso lo suficientemente representativo para permitir al sistema dar estimaciones con el mayor grado de precisión posible.

Para ello, cada escena generada automáticamente es sometida a un proceso de bootstrapping cuyo resultado será un gran número de configuraciones de render diferentes para la misma escena. Ningún aspecto de la geometría del modelo es alterado, tan sólo los parámetros del proceso de render que se lanzarán cada vez sobre el modelo.

[2]En el ámbito de ciencias de la computación, la palabra inglesa *bootstrap* hace referencia al hecho de crear un sistema complejo a través de uno más simple.

A la hora de definir cómo se alteran los parámetros de render, se han identificado dos posibles aproximaciones. La primera consiste en tratar de conseguir configuraciones similares a las utilizadas normalmente por los usuarios. Una posible forma de conseguir esto sería haciendo pequeñas variaciones aleatorias sobre un conjunto de configuraciones realizadas por usuarios expertos (*bootstrapping evolutivo*). La segunda aproximación consiste en crear configuraciones completamente aleatorias (*bootstrapping aleatorio*). Para ello, a cada atributo definido en 4.2.1 se le asigna un valor aleatorio dentro del rango de valores permitidos por cada atributo en particular.

Ambas aproximaciones tienen ventajas y desventajas. En primer lugar, el bootstrapping evolutivo tiene la ventaja de obtener configuraciones con sentido desde el punto de vista del usuario experto con una mayor probabilidad, mientras que su desventaja principal es que las configuraciones generadas están sesgadas por el criterio del usuario que ha definido la configuración inicial. Por otro lado, el bootstrapping aleatorio tiene el inconveniente de dar lugar a configuraciones que carecen de sentido, pero dichas configuraciones no están sesgadas, ya que son totalmente aleatorias.

La aproximación utilizada a la hora de realizar el presente proyecto es la que consiste en *bootstrapping aleatorio*. La razón de tal decisión se debe a que aún con el problema de generar configuraciones de mala calidad, esto es preferible sobre la posibilidad de generar una muestra de ejemplares sesgada por algún criterio. No obstante, las malas configuraciones de render darán lugar a resultados con un valor de calidad muy bajo, con lo que serán discriminadas automáticamente por el sistema.

4.2.3. Render de instancias

El objetivo de esta etapa es llevar a cabo el render de las escenas de la base de conocimiento con las configuraciones de parámetros generadas en la etapa anterior. Como resultado, se obtendrán el tiempo de render empleado por cada configuración sobre cada modelo y las imágenes generadas.

Todas las instancias del proceso de render realizadas para obtener la base de conocimiento se han llevado a cabo en un computador de sobremesa con las siguientes características:

- Blender v2.45
- Yafray v0.0.9
- Python v2.4.4
- Sistema operativo Debian GNU/Linux
- Linux v2.6.22
- Procesador AMD Athlon XP 2000+ 2.0GHz
- Memoria principal de 1024 MB DDR 433Hz

4.2.4. Evaluación de la calidad de los resultados

El objetivo de esta etapa es evaluar la calidad de las distintas imágenes generadas por las diferentes configuraciones de render lanzadas sobre un mismo modelo 3D. Así es posible etiquetar la calidad de las diferentes instancias del proceso de render lanzadas sobre una misma escena.

A diferencia del tiempo, la calidad de una imagen es un criterio subjetivo. Normalmente, los resultados deseados son imágenes con un buen contraste, brillo y saturación, y sin manchas provocadas por *ruido* en el proceso. Esto plantea un nuevo problema: ¿Cómo se puede evaluar de una forma automática la calidad de un conjunto de imágenes generadas?

Suponiendo un conjunto I de n imágenes, la aproximación tomada para resolver el problema consiste en tomar una imagen i_m de referencia seleccionada por el usuario como la *mejor* imagen. A partir de esta imagen, todas las demás del conjunto serán evaluadas según sus respectivas distancias con la imagen i_m.

Medida de distancia entre imágenes

La medida de distancia utilizada para evaluar cómo de similares (o diferentes) son dos imágenes es la *RMSD*[3]. Esta medida da una aproximación de la diferencia entre dos imágenes y se define como:

[3]Del inglés *Root Mean Square Deviation*

$$RMSD(I_1, I_2) = \sqrt{MSE(I_1, I_2)} = \sqrt{\frac{\sum_{i=1}^{n}(p_{1,i} - p_{2,i})^2}{n}} \tag{4.1}$$

donde I_1 e I_2 son las imágenes a comparar y p_1 y p_2 son sus respectivos conjuntos de píxeles. Así, esta medida será cero cuando se comparen dos imágenes idénticas y un valor positivo directamente proporcional al grado de diferencia al comparar imágenes distintas. Podemos decir por lo tanto que esta medida da una aproximación del grado de diferencia entre dos imágenes.

Con la medida de distancia entre dos imágenes definida y a partir de una imagen de referencia I_{ref} tomada como la *mejor* del conjunto, es posible asignar un valor de calidad a cada imagen I_i inversamente proporcional a la distancia $D(I_{ref}, I_i)$ entre cada imagen y la imagen de referencia. Así, si la escala de calidad contiene valores en el intervalo $[0, 10]$, donde 10 es la mayor calidad y 0 la menor, las imágenes más similares a la imagen de referencia tendrán calidad más cercana al 10, mientras que las más diferentes la tendrán cercana al 0.

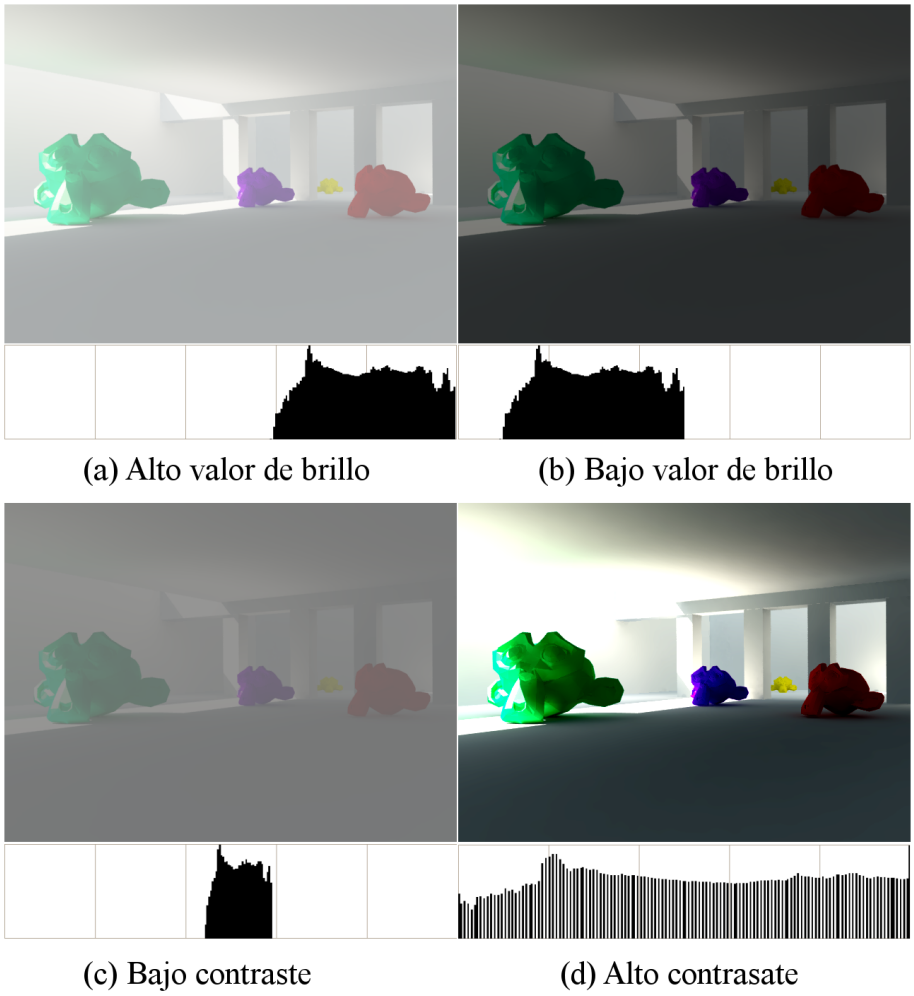
La $RMSD$ es una medida de la desviación de un conjunto de valores sobre otro. Los valores en nuestro caso son valores de píxeles, es decir, el color de la imagen. Así, esta medida da una aproximación al grado de diferencia entre dos imágenes basado en la diferencia de color de cada píxel. Esto es un problema en ciertos casos. Supongamos una imagen de referencia I_{ref} con unos ciertos valores de contraste, brillo y saturación. Sean ahora dos imágenes diferentes I_1 e I_2 a evaluar, donde I_1 es de muy buena calidad, pero tiene valores de brillo, saturación y contraste muy diferentes a los de I_{ref}, e I_2, aunque no es de tan buena calidad, tiene unos valores de brillo, saturación y contraste muy similares a los de I_{ref}. En este caso, $RMSD(I_{ref}, I_1) > RMSD(I_{ref}, I_2)$ dando lugar a una evaluación incorrecta en la que a I_2 se le asigna una mayor calidad que a I_1.

Tratamiento de imagen mediante la manipulación de histogramas

Para solucionar el problema mencionado, se debe preprocesar cada imagen a comparar tratando de ajustar sus valores de brillo, contraste y saturación a los de la imagen de referencia. Para ello se introduce el método de *proyección de histograma*[4] [17].

[4]Traducción del término inglés *histogram matching.*

Sea I una imagen codificada con niveles de gris en el rango $[0, L-1]$, el histograma de I es una función $h(r_k) = n_k$, donde r_k es el k nivel de gris y n_k es el número de píxeles en la imagen cuyo valor es r_k[5].



(a) Alto valor de brillo (b) Bajo valor de brillo

(c) Bajo contraste (d) Alto contrasate

Figura 4.7: Efectos de la manipulación de histograma sobre una misma imagen.

Los histogramas son muy útiles en el campo de procesamiento y tratamiento de imagen. Además de proporcionar estadísticas útiles para ciertas aplicaciones como la compresión de imagen, son la base de numerosas técnicas de manipulación de imagen. En la figura 4.7 puede verse el efecto de manipular las características de una misma imagen sobre su histograma. En 4.7.(a) puede verse cómo una imagen con un alto valor de brillo tiene un histograma que

[5]Por simplicidad se ha supuesto una imagen en escala de grises, aunque esta definición es generalizable a imágenes a color con más de un solo canal.

tiende a concentrar más píxeles en los valores altos (más hacia la derecha) del histograma. Por otro lado, en 4.7.(b) puede verse cómo una imagen oscura tiende a concentrar más píxeles en la parte izquierda del histograma (la correspondiente a los valores más bajos de color). En 4.7.(c) puede verse una imagen con bajo contraste cuyo histograma tiende a concentrar todos sus píxeles en unos pocos valores centrales del histograma. Por último, en 4.7.(d) se observa cómo una imagen de alto contraste tiene un histograma cuyos píxeles tienden a ocupar todos los posibles valores de color y además la distribución de estos es casi uniforme.

Ecualización de histograma

La primera técnica de manipulación automática de histograma es la *ecualización de histograma*. Esta técnica trata de distribuir el histograma de una imagen de forma uniforme a través de todos los posibles valores de píxeles. Esto da lugar a imágenes con un alto contraste. Esta técnica es muy útil en casos de imágenes con histogramas muy concentrados en un rango muy pequeño de valores, ya que puede ayudar a revelar información de la imagen que permaneció oculta debido a la oscuridad o luminosidad excesiva.

Consideremos para esta explicación funciones continuas y que la variable r representa los niveles de gris de una imagen de un sólo canal (nivel de grises) a ser tratada, aunque puede ser extrapolada a imágenes con más de un solo canal de color. Además, se asume que r ha sido normalizada en el intervalo $[0, 1]$, con $r = 0$ representando el color negro y $r = 1$ el blanco. Para realizar la ecualización de histograma, necesitamos transformaciones de la forma

$$s = T(r) \quad 0 \leq r \leq 1 \tag{4.2}$$

que produce un nivel de color s para cada valor de píxel r de la imagen original. Por razones que serán obvias posteriormente en esta explicación, se asume que la función $T(r)$ satisface las siguientes condiciones:

1. $T(r)$ toma valores únicos y es monótona creciente en el intervalo $0 \leq r \leq 1$
2. $0 \leq T(r) \leq 1$ para $0 \leq r \leq 1$

En la condición *a)*, $T(r)$ debe tomar valores únicos para garantizar que la transformación inversa existe y el incremento monótono garantiza que exista un orden creciente desde el

negro hasta el blanco. La condición *b)* garantiza que los niveles de color de salida están en el mismo rango que los de entrada. La transformación inversa desde s a r se define como

$$r = T^{-1}(s) \quad 0 \leq s \leq 1 \tag{4.3}$$

Los niveles de gris de una imagen pueden ser vistos como variables aleatorias en el intervalo $[0, 1]$. Uno de los descriptores más importantes para variables aleatorias es la función de densidad de probabilidad (FDP). Sean $p_r(r)$ y $p_s(s)$ las funciones de densidad de probabilidad de las variables aleatorias r y s respectivamente, si $p_r(r)$ y $T(r)$ son conocidas y $T^{-1}(s)$ satisface la condición *a)*, entonces la FDP $p_s(s)$ de la variable s puede ser obtenida según (4.4).

$$p_s(s) = p_r(r) \left| \frac{dr}{ds} \right| \tag{4.4}$$

Así, la FDP de una variable s es determinada por la FDP del nivel de grises de la imagen de entrada y por la función de transformación escogida.

Una función de transformación particularmente importante en tratamiento de imagen es:

$$s = T(r) = \int_0^r p_r(w)dw \tag{4.5}$$

donde w es una variable de conveniencia para la integración. La parte derecha de (4.5) es la función de distribución acumulada (FDA) de la variable aleatoria r. Ya que las FDP son siempre positivas y recordando que la integral de una función es el área bajo la función, la expresión (4.5) satisface la condición *a)* al ser simplemente valuada y monótona creciente. La condición *b)* también es satisfecha al estar la integral de una función de densidad de probabilidades para variables en el rango $[0, 1]$ también en dicho rango.

Dada $T(r)$, podemos calcular $p_s(s)$ aplicando la expresión (4.4).

$$\frac{ds}{dr} = \frac{dT(r)}{dr} = \frac{d}{dr}\left[\int_0^r p_r(w)dw\right] = p_r(r) \tag{4.6}$$

Por lo que, sustituyendo en (4.4) y teniendo en cuenta que todo valor de probabilidad es positivo:

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| = p_r(r)\left|\frac{1}{p_r(r)}\right| = 1 \quad 0 \leq s \leq 1 \tag{4.7}$$

La expresión (4.7) recibe el nombre de función de densidad de probabilidad uniforme.

En el caso de valores discretos, la demostración realizada anteriormente es representada en base a probabilidades y sumas en lugar de funciones de densidad de probabilidad e integrales. Así, la probabilidad de la ocurrencia de un nivel de gris r_k en una imagen es aproximada por

$$p_r(r_k) = \frac{n_k}{n} \quad k = 0, 1, 2, ..., L-1 \tag{4.8}$$

donde n es el número total de píxeles en la imagen, n_k es el número de píxeles que tienen un valor de gris r_k y L es el número total de niveles de gris posibles en la imagen. La versión discreta de la expresión (4.5) es

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} \frac{n_j}{n} \quad k = 0, 1, 2, ..., L-1 \tag{4.9}$$

Por lo tanto, una imagen procesada es obtenida proyectando cada píxel de la imagen de entrada con un nivel de color r_k en un píxel con nivel de color s_k de la imagen procesada mediante la expresión (4.9).

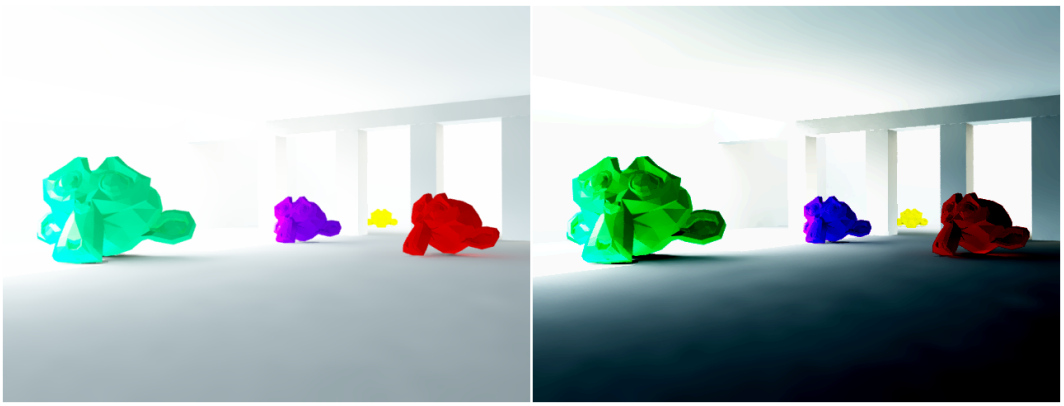


Figura 4.8: Efecto de aplicar el método de ecualización de histograma sobre una imagen: **Izquierda**: Imagen original, **Derecha**: Imagen ecualizada (ver versión a color en C.3).

Esta transformación recibe el nombre de *ecualización de histograma*. Aunque la versión discreta de esta transformación no llega a producir los resultados de su contrapartida continua,

la expresión (4.9) tiende a expandir el histograma de la imagen de entrada a través de todos los posibles niveles de color, con lo que se consigue una buena aproximación. La figura 4.8 muestra una imagen con un alto valor de luminosidad (izquierda) y la imagen resultado de aplicar la ecualización de histograma sobre ella (derecha).

Proyección de histograma

La ecualización de histograma es una técnica muy útil, ya que permite obtener resultados predecibles de forma automática y es un método muy sencillo de implementar. Aún así, hay casos en los que la obtención de un histograma con distribución uniforme no es la mejor aproximación para tratar una imagen. En estos casos puede ser interesante especificar qué forma debe tener el histograma de la imagen resultado. El método utilizado para generar una imagen con un histograma específico se llama *proyección de histograma*.

Consideremos de nuevo los niveles de gris continuos r y z como variables aleatorias continuas y sean $p_r(r)$ y $p_z(z)$ sus correspondientes funciones de densidad de probabilidad. Las variables r y z denotan los niveles de gris de las imágenes de entrada y salida, respectivamente. Mientras $p_r(r)$ se puede estimar directamente a partir de la imagen de entrada, $p_z(z)$ es la función de distribución de probabilidad específica que queremos que tenga la imagen procesada.

Volviendo por un momento a la expresión (4.5), donde definimos la variable aleatoria s, supongamos que definimos una variable aleatoria z con la siguiente propiedad

$$G(z) = \int_0^z P_z(t)dt = s \tag{4.10}$$

donde t es una variable de conveniencia para la integración.

De las expresiones (4.5) y (4.10) se deduce que $G(z) = T(r)$ y, por lo tanto, z debe satisfacer la siguiente expresión

$$z = G^{-1}(s) = G^{-1}\left[T(r)\right] \tag{4.11}$$

La función de transformación $T(r)$ puede obtenerse a partir de la expresión (4.5) una vez $p_r(r)$ ha sido estimada a partir de la imagen de entrada. Similarmente, la función de transfor-

mación $G(z)$ puede ser obtenida a partir de la expresión (4.10), ya que $p_z(z)$ es conocida.

Asumiendo que G^{-1} existe y satisface las condiciones (a) y (b) expuestas en el apartado anterior, estas expresiones muestran que es posible obtener una imagen con una función de distribución de probabilidad específica mediante el siguiente procedimiento:

1. Obtener la función de transformación $T(r)$ mediante la ecuación (4.5).
2. Obtener la función de transformación $G(z)$ mediante la ecuación (4.10).
3. Obtener la función de transformación G^{-1}.
4. Obtener la imagen resultado aplicando la expresión (4.11) a cada píxel en la imagen procesada.

El resultado de este procedimiento será una imagen cuyos niveles de gris tienen una función de distribución de probabilidad específica.

Como en el caso de la ecualización de histograma, la contrapartida discreta de esta explicación es el método utilizado para implementar este procedimiento en un programa de computador. Sin embargo, la variante discreta permite la implementación fácil de este procedimiento a cambio de la obtención de una aproximación al resultado deseado.

La variante discreta de la expresión (4.5) es definida en (4.9) y repetida aquí por comodidad en la lectura

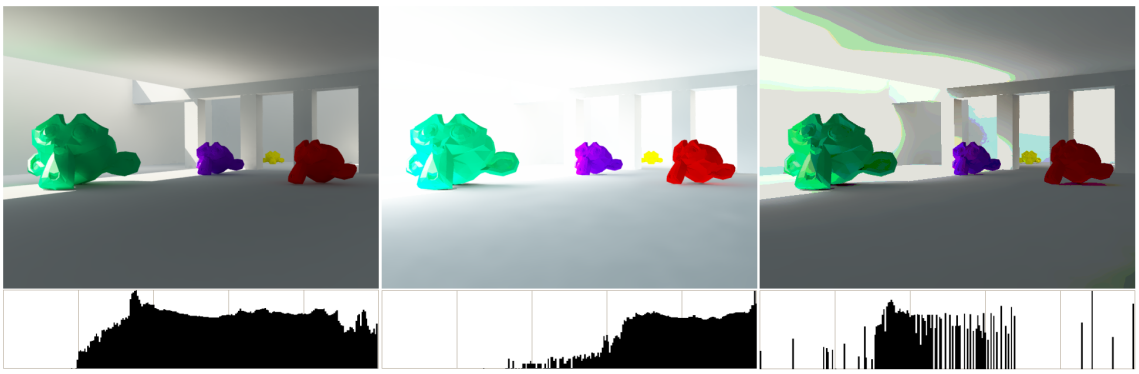
$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} \frac{n_j}{n} \quad k = 0, 1, 2, ..., L-1 \tag{4.12}$$

donde n es el número total de píxeles en la imagen, n_j es el número de píxeles con un nivel de color r_j y L es el número total de niveles de color. De forma similar, la variante discreta de la expresión (4.10) es obtenida a partir del histograma definido por $p_z(z_i)$ para todo $i = 0, 1, 2, ..., L-1$

$$v_k = G(z_k) = \sum_{i=0}^{k} p_z(z_i) = s_k \quad k = 0, 1, 2, ..., L-1 \tag{4.13}$$

Finalmente, la versión discreta de la expresión (4.11) es dada por

$$z_k = G^{-1}(s_k) \quad k = 0, 1, 2, ..., L-1 \tag{4.14}$$



(a) Imagen de referencia (b) Imagen a comparar (c) Imagen resultado

Figura 4.9: Ejemplo de imagen procesada mediante proyección de histograma (ver versión a color en C.4)

En la figura 4.9 puede verse un ejemplo de imagen procesada mediante proyección de histograma. En 4.9.(a) se puede ver la imagen cuyo histograma deseamos obtener en la imagen resultado. La imagen 4.9.(b) es la imagen a procesar. El resultado de aproximar el histograma de (b) al de (a) es la imagen 4.9.(c). Como se puede notar en este caso particular, cuanto más diferentes sean los histogramas de la imagen de entrada y la de referencia, peor será la aproximación realizada. Aún así, el resultado conseguido permite solucionar de forma suficientemente aceptable el problema mencionado ajustando los niveles de brillo, saturación y contraste de una imagen a los de otra tomada como referencia.

4.2.5. Extracción y codificación de información

En esta etapa ya tenemos disponible toda la información necesaria para representar las instancias de render que serán pasadas al núcleo de aprendizaje del sistema. El objetivo de esta etapa es agrupar toda esta información y representarla en un formato que sea *entendible* por el núcleo de aprendizaje.

Una de las formas más comunes de representar la información en los diferentes paquetes de software de estadística, aprendizaje máquina y minería de datos es el formato de fichero

tabular. En este, la información se codifica en un fichero de texto plano y se representa como una tabla cuyas filas corresponden a las instancias y columnas a los diferentes atributos que describen dichas instancias. El carácter delimitador de fila es el de *nueva línea*, mientras que el de columna es el *tabulador*. Además, en este formato la primera fila está reservada para la descripción de los atributos del dominio. El formato de fichero soportado por YAReW sigue estas especificaciones, aunque con algunas particularidades en la cabecera que describe los atributos del dominio.

La primera línea de un fichero de conjunto de instancias es la línea de descripción del dominio. En esta se especifican los nombres de cada atributo separados por tabuladores. Por conveniencia, el último atributo definido será la clase o atributo objetivo del dominio. La descripción de los tipos de datos correspondientes se puede hacer bien de forma automática, dejando al núcleo del sistema que decida qué tipo de datos corresponde a cada atributo, o bien de forma explícita, especificando los diferentes tipos de datos mediante prefijos en los nombres de los atributos.

Cuando la línea de descripción del dominio contiene los nombres de los atributos sin ningún prefijo *entendible* por el núcleo del sistema, este trata de asignar de forma automática los tipos de datos más convenientes para cada atributo observando los valores que estos toman a través de cada instancia del modelo. Para ello, se sigue la siguiente convención:

- Atributos cuyos valores son dígitos entre 0 y 9 o algún subconjunto de estos son tomados como discretos.
- Atributos cuyos valores pueden ser *parseados* como números y no están definidos en el intervalo $[0, 9]$ son tomados como continuos.
- Atributos cuyos valores son cadenas de texto son tomados como discretos.
- El último atributo definido es tomado como la clase del dominio.

Cuando este procedimiento no es apropiado para describir un dominio dado, es posible utilizar prefijos en los nombres de los atributos para especificar al sistema qué tipos de datos debe asignar a cada uno de ellos. Los prefijos pueden tener una o dos letras seguidas del carácter #. El primer carácter del prefijo es utilizado para especificar el atributo objetivo del

dominio y es el carácter *c*. Sólo es posible especificar un atributo objetivo o clase en un dominio dado, por lo que sólo un atributo puede tener el carácter *c* en su prefijo. El segundo carácter del prefijo es utilizado para especificar el tipo de datos del atributo en cuestión y puede ser *D* para especificar un atributo discreto o *C* para continuo.

A continuación puede verse un ejemplo de fichero de datos en el cuál se ha optado por una descripción del dominio automática por parte del sistema. En este se definen cuatro atributos: *sepal length*, *sepal width*, *petal length* y *petal width*. Ya que los valores que toman las diferentes instancias en dichos atributos son todos números reales, el sistema tomará dichos atributos como continuos. El último atributo, *iris*, será la clase del dominio. Ya que los valores que toman las instancias sobre este atributo son cadenas de texto, el sistema asignará un tipo de datos discreto sobre él.

```
sepal length sepal width petal length petal width iris

5.1 3.5 1.4 0.2 Iris-setosa
4.9 3.0 1.4 0.2 Iris-setosa
4.7 3.2 1.3 0.2 Iris-setosa
4.6 3.1 1.5 0.2 Iris-setosa
5.0 3.6 1.4 0.2 Iris-setosa
5.7 3.0 4.2 1.2 Iris-versicolor
5.7 2.9 4.2 1.3 Iris-versicolor
6.2 2.9 4.3 1.3 Iris-versicolor
5.1 2.5 3.0 1.1 Iris-versicolor
5.7 2.8 4.1 1.3 Iris-versicolor
6.3 3.3 6.0 2.5 Iris-virginica
5.8 2.7 5.1 1.9 Iris-virginica
7.1 3.0 5.9 2.1 Iris-virginica
6.3 2.9 5.6 1.8 Iris-virginica
6.5 3.0 5.8 2.2 Iris-virginica
7.6 3.0 6.6 2.1 Iris-virginica
```

El ejemplo siguiente es el mismo modelo especificado en el ejemplo anterior, salvo que esta vez se ha optado por hacer una descripción explícita de el mismo. Como puede observarse, los cuatro atributos descriptivos, *sepal length*, *sepal width*, *petal length* y *petal width*, comparten el prefijo *C#*, lo que especifica al sistema que debe tratar dichos atributos como continuos. Por último, el atributo *iris* tiene un prefijo *cD#*, lo que especifica al sistema que dicho atributo es la clase del dominio (el carácter *c*) y además es de naturaleza discreta (carácter *D*).

```
C#sepal length C#sepal width C#petal length C#petal width cD#iris
```

```
5.1 3.5 1.4 0.2 Iris-setosa
4.9 3.0 1.4 0.2 Iris-setosa
4.7 3.2 1.3 0.2 Iris-setosa
4.6 3.1 1.5 0.2 Iris-setosa
5.0 3.6 1.4 0.2 Iris-setosa
5.7 3.0 4.2 1.2 Iris-versicolor
5.7 2.9 4.2 1.3 Iris-versicolor
6.2 2.9 4.3 1.3 Iris-versicolor
5.1 2.5 3.0 1.1 Iris-versicolor
5.7 2.8 4.1 1.3 Iris-versicolor
6.3 3.3 6.0 2.5 Iris-virginica
5.8 2.7 5.1 1.9 Iris-virginica
7.1 3.0 5.9 2.1 Iris-virginica
6.3 2.9 5.6 1.8 Iris-virginica
6.5 3.0 5.8 2.2 Iris-virginica
7.6 3.0 6.6 2.1 Iris-virginica
```

4.2.6. Evaluación del modelo

La última etapa en la construcción de la base de conocimiento es la evaluación del modelo obtenido. Las entradas a esta etapa son dos ficheros de conjuntos de datos, en los cuales se encuentran todas las instancias de render generadas y proyectadas sobre el dominio del tiempo y sobre el de la calidad. El objetivo es, por tanto, evaluar cómo de apropiado es el modelo obtenido para inducir hipótesis que describan bien los datos y que permitan estimar, lo más precisamente posible, nuevas instancias de render que no hayan sido utilizadas en el entrenamiento del algoritmo de aprendizaje.

Evaluar un modelo de datos desde este punto de vista no es una tarea fácil. Según el algoritmo de aprendizaje utilizado, el modelo de datos dará lugar a la construcción de una hipótesis u otra, por lo que el objetivo es buscar aquel modelo de datos que dé lugar a la construcción de hipótesis que generalicen de la mejor forma posible independientemente del algoritmo de aprendizaje utilizado.

En este proyecto en concreto, el método de evaluación empleado ha sido *validación cruzada* en su variante de diez hojas. Por consiguiente, el proceso de evaluación ha sido dividido en diez iteraciones cada una de ellas formada por un 90 % de los datos destinados a entrenar el algoritmo de aprendizaje y el 10 % restante para validarlo. Cada conjunto de instancias generadas deben ser proyectadas sobre el dominio de tiempo y el de calidad. Esto significa que se deben utilizar dos algoritmos distintos, uno de clasificación para estimar la calidad y otro

de regresión para el tiempo. Por consiguiente, para evaluar la actuación de cada algoritmo se deben utilizar medidas distintas. La proporción de instancias clasificadas correctamente será la medida de rendimiento del algoritmo de clasificación empleado, mientras que el error medio al cuadrado será la medida en el caso del algoritmo de regresión.

Los algoritmos utilizados han sido árboles de decisión (de clasificación en el caso de estimar calidad y de regresión en el caso del tiempo) y *k* vecinos más cercanos con un valor de $k = 8$.

Al finalizar esta etapa, puede considerarse volver a la etapa de definición de atributos, al principio del proceso, para incluir o eliminar ciertos atributos y observar cómo afectan los cambios en el dominio a la calidad de las hipótesis inducidas a partir del modelo.

4.3. Desarrollo de YAReW

4.3.1. Arquitectura del sistema

YAReW ha sido desarrollado de acuerdo a una arquitectura de componentes definida con el fin de independizar las posibles formas que puedan tomar las entradas y la forma de visualización de los resultados del núcleo de aprendizaje del sistema. Los componentes que forman el sistema están organizados en tres niveles o etapas diferentes. En la figura 4.10 puede verse un esquema de la arquitectura propuesta. A continuación se describen brevemente los niveles que la forman.

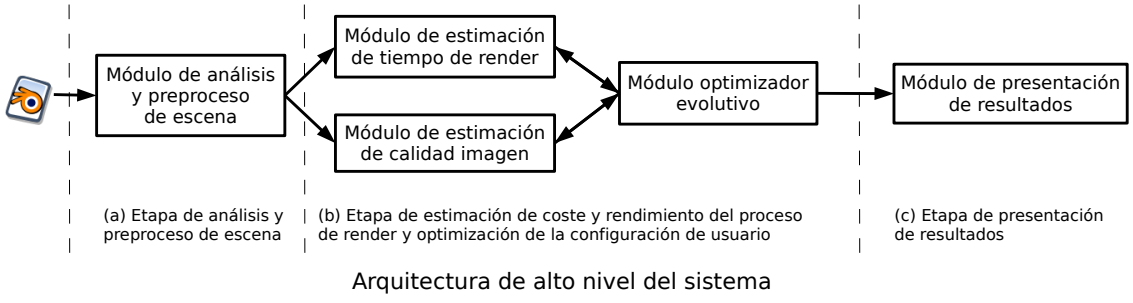


Figura 4.10: Arquitectura del sistema propuesto

- **Etapa de análisis y preproceso de la escena**: En esta etapa, el modelo proporcionado por el usuario es analizado con el objetivo de identificar el modelo de la base de conocimiento más similar al modelo de usuario. El fin de este procedimiento es la utilización del conjunto de datos derivado del modelo de la base de conocimiento más similar al modelo de usuario para poder dar estimaciones de tiempo de render y calidad de resultados lo más aproximadamente posible.

- **Etapa de estimación de coste y rendimiento del proceso de render y optimización de la configuración de usuario**: Esta etapa recibe como entrada la tupla de atributos que caracteriza al modelo de usuario preparada y formateada de acuerdo con los requisitos del motor de aprendizaje del sistema. El objetivo de esta etapa es estimar el tiempo de ejecución de una instancia del proceso de render y la calidad del resultado obtenido según la configuración de parámetros de entrada al proceso establecida por el usuario. Así mismo, en un paso posterior dentro de la misma etapa, el módulo de optimización evolutiva tratará de optimizar la configuración establecida por el usuario con el objetivo de maximizar la relación $\frac{\text{Calidad de Resultado}}{\text{Tiempo de Render}}$.

- **Etapa de presentación de resultados**: Esta etapa toma como entradas las estimaciones de tiempo y resultados realizadas sobre el modelo de usuario y la configuración de parámetros óptima según el sistema. El objetivo aquí es mostrar de una forma conveniente toda esta información al usuario.

4.3.2. Etapa de análisis y preproceso de la escena de usuario

Este nivel tiene como objetivo analizar el modelo 3D proporcionado por el usuario para encontrar aquel modelo de la base de conocimiento que más se asemeje a él. Además, con esta etapa se consigue una independencia entre el formato de representación de los modelos de usuario (entradas al sistema) con respecto al núcleo de aprendizaje, ya que los detalles acerca de la configuración de render establecida por el usuario son extraídos y representados en un formato conveniente para ser procesados por el núcleo del sistema.

Las entradas en esta etapa son un modelo 3D y la definición de una configuración de parámetros del proceso de render proporcionados por el usuario. Las salidas de esta son una

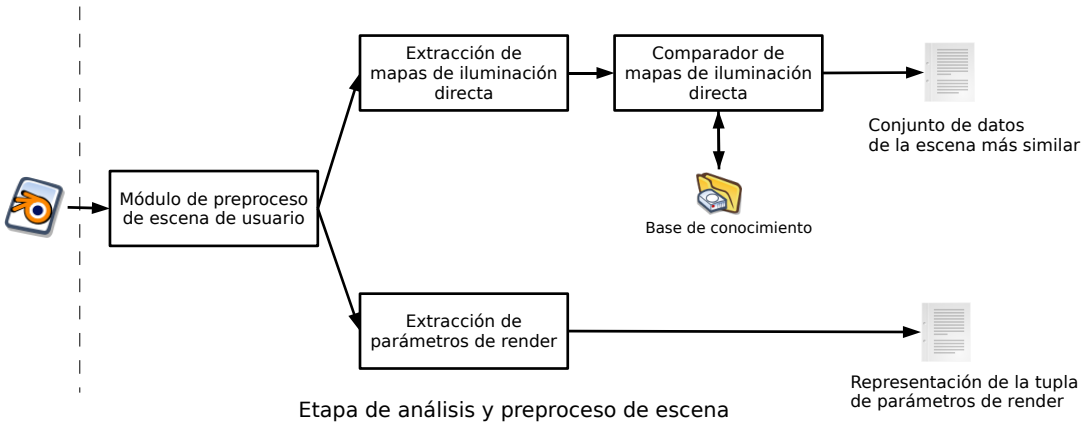


Figura 4.11: Etapa de análisis y preproceso de escena de usuario

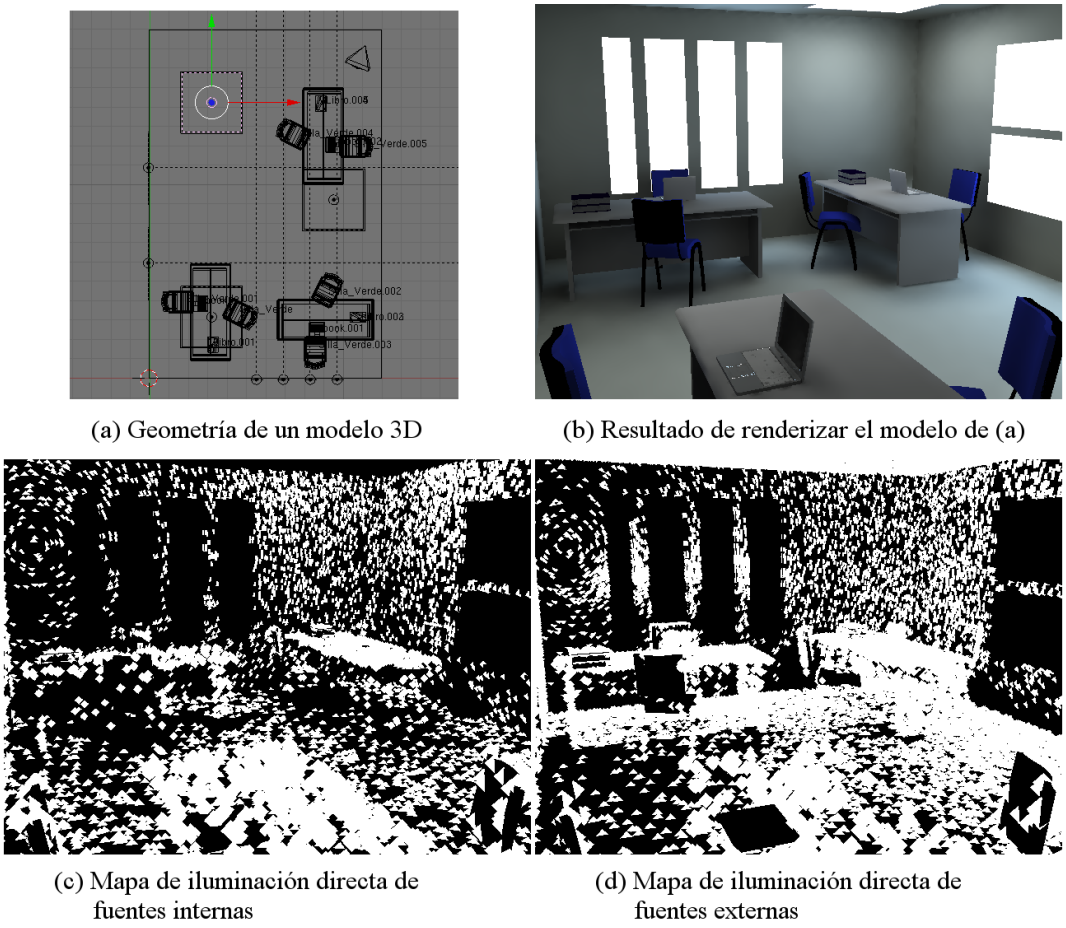
tupla de atributos que caracteriza la configuración de render propuesta por el usuario y el conjunto de datos derivado del modelo de la base de conocimiento del sistema más similar al modelo de usuario. En la figura 4.11 puede verse un esquema de la estructura de este nivel. A continuación se describe qué son los mapas de iluminación directa, cómo se extraen y para qué son utilizados. Posteriormente, se detallará cómo es extraída la información necesaria de una escena para dar lugar a una instancia de render lista para su utilización por el núcleo de aprendizaje.

Mapas de iluminación directa

El primero de los objetivos de esta etapa es identificar el modelo de la base de conocimiento que se asemeje en un mayor grado al proporcionado por el usuario. Esto se consigue gracias a un artefacto denominado **mapas de iluminación directa**. Los mapas de iluminación directa son imágenes bidimensionales que representan los impactos de la luz provenientes directamente de las fuentes de iluminación definidas en la escena. Por lo tanto, estos artefactos aportan una estimación al modelo de iluminación de una escena como consecuencia de la acción directa de las fuentes de iluminación definidas en ella.

A partir de los impactos de iluminación directa sobre los objetos de la escena, se generan nuevos impactos que rebotan entre los distintos objetos, dando así lugar a iluminación indi-

recta. Tanto la iluminación directa, como la indirecta tienen un gran impacto en el tiempo de ejecución de los algoritmos de iluminación global y en la calidad de los resultados producidos. Por este motivo, la medida de distancia utilizada para comparar el modelo proporcionado por el usuario con los contenidos en la base de conocimiento serán estos artefactos.



(a) Geometría de un modelo 3D

(b) Resultado de renderizar el modelo de (a)

(c) Mapa de iluminación directa de fuentes internas

(d) Mapa de iluminación directa de fuentes externas

Figura 4.12: Mapas de iluminación directa de una escena generada automáticamente

Recordando, las escenas de interés en este estudio son escenas de interiores, donde hay definidas fuentes de iluminación internas a la escena, como lámparas o bombillas, y fuentes externas, como iluminación solar que entra a escena a través de las ventanas. Para estimar el modelo de iluminación de la escena se tienen en cuenta ambos tipos de fuentes, aunque se toman por separado. Así, para una escena dada se obtiene un mapa de iluminación directa interna y otro de iluminación directa externa. En la figura 4.12 puede verse una escena generada automáticamente (ver generador de escenas en 4.2) y sus mapas de iluminación directa

de fuentes internas y externas.

Extracción de mapas de iluminación directa

La extracción de mapas de iluminación de una escena es un proceso no trivial y muy costoso en términos computacionales. En él interviene un proceso de trazado de rayos simplificado que simula el comportamiento de la luz sobre la geometría de la escena. En la figura 4.13 puede verse un diagrama de flujo de este proceso. A continuación se describe cada etapa que lo compone.

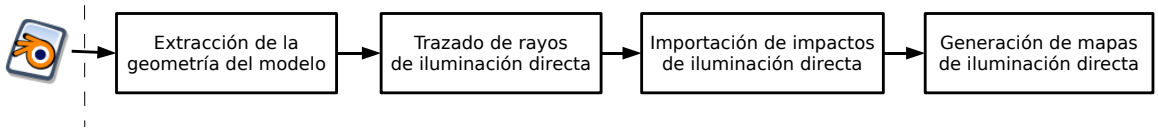


Figura 4.13: Proceso de extracción de los mapas de iluminación directa de una escena

La primera etapa en este proceso es la **extracción de la geometría del modelo**. El objetivo de esta etapa es la extracción de toda la información de geometría de la escena de usuario que es relevante para el trazador de impactos. Esta información está formada por un conjunto de polígonos, un conjunto de fuentes de iluminación y una cámara virtual. Cada uno de estos elementos debe estar descrito con un nivel de detalle suficiente que permita al trazador de impactos reconstruir la geometría de la escena de forma precisa. A continuación se describe la información requerida de estos elementos.

- **Conjunto de polígonos definidos en la escena**: Todos los objetos de tipo *malla* definidos en la escena están formados por polígonos. Por razones de simplicidad y eficiencia, el trazador de impactos requiere polígonos de tres vértices, por lo que cada polígono en la escena debe ser descompuesto en triángulos. Cada triángulo es exportado definiendo sus tres vértices, descritos mediante coordenadas cartesianas, y su vector normal.

- **Conjunto de fuentes de iluminación**: Cada fuente de iluminación es descrita especificando su nombre, punto de localización, tipo de fuente, vector normal y energía de

iluminación. El nombre es necesario para identificar cada fuente de iluminación como interna o externa a la escena. Por convenio, toda fuente de iluminación definida en el interior de la escena debe comenzar su nombre con la cadena "*int*". Del mismo modo, toda fuente externa a la escena debe comenzar su nombre con la cadena "*ext*". Cualquier fuente cuyo nombre no comience con cualquiera de estas cadenas será ignorada por el trazador de impactos y por consiguiente su efecto será ignorado en el mapa de iluminación directa obtenido[6]. El punto de localización de una fuente de iluminación es el origen de los rayos de luz que esta emite. El tipo de fuente es necesario, ya que cada tipo tiene un comportamiento diferente a la hora de distribuir rayos de luz. En casos de fuentes con un área definida, el vector normal del plano correspondiente es necesario. Por último, la energía de iluminación es necesaria, ya que aquellos impactos que provengan de fuentes de iluminación con mayor energía serán más visibles que aquellos que provengan de fuentes con menor energía.

- **Cámara virtual**: La cámara virtual de un modelo 3D es el punto de vista desde el que se produce el renderizado de la escena. La información necesaria acerca de la cámara es su posición y coordenadas del vector normal, entendido como el vector hacia donde la cámara está enfocada.

Toda esta información es representada en un fichero de intercambio en un formato entendible por el trazador de impactos. A continuación puede observarse un ejemplo de dicho fichero.

```
#################################################################
# 3D InterXange File Format
#
#################################################################
# CAMERA: Position(x,y,z)  | Look  (Vector) |  Name
c 11.151 17.176 4.716 -0.362 -0.912 -0.190 Camera
# LAMP:  Name | Location(x,y,z) | Type | Look (Vector) | Light intensity
l intLamp 9.525 9.225 6.936 Area 0.000 0.000 -1.000 1.000
# LAMP:  Name | Location(x,y,z) | Type | Look (Vector) | Light intensity
l intLamp.001 3.219 14.245 6.936 Area 0.000 0.000 -1.000 1.000
# LAMP:  Name | Location(x,y,z) | Type | Look (Vector) | Light intensity
l extLamp -0.050 5.956 2.535 Area 1.000 -0.000 0.000 1.000
```

[6]Se ha optado por esta solución debido a la gran dificultad que conlleva realizar una identificación automática ambos tipos de fuentes.

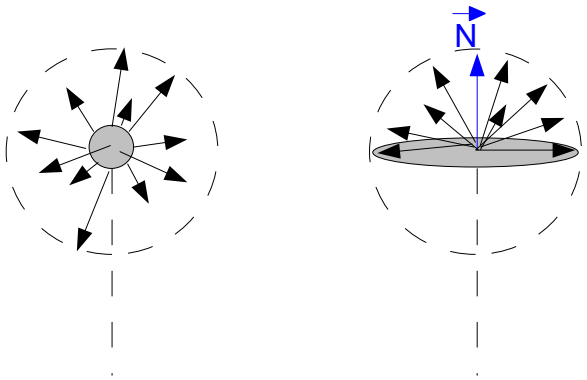
```
# LAMP:  Name | Location(x,y,z) | Type | Look (Vector) | Light intensity
l extLamp.001 -0.050 10.882 2.535 Area 1.000 -0.000 0.000 1.000
# LAMP:  Name | Location(x,y,z) | Type | Look (Vector) | Light intensity
l extLamp.002 -0.050 5.956 4.837 Area 1.000 -0.000 0.000 1.000
# TRIANGLE V0x V0y V0z | V1x V1y V1z | V2x V2y V2z | Normal (Nx Ny Nz)
t 2.899 3.623 1.999 2.899 3.665 1.991 2.899 3.665 1.959 -1.000 0.000 0.000
t 2.899 3.623 1.991 2.899 3.665 1.959 2.899 3.623 1.959 -1.000 0.000 0.000
t 2.258 3.665 1.991 2.258 3.623 1.991 2.258 3.623 1.959 1.000 -0.000 0.000
t 2.258 3.665 1.991 2.258 3.623 1.959 2.258 3.665 1.959 1.000 -0.000 0.000
t 2.899 3.665 1.991 2.899 3.735 1.991 2.899 3.735 1.959 -1.000 0.000 0.000
t 2.899 3.665 1.991 2.899 3.735 1.959 2.899 3.665 1.959 -1.000 0.000 0.000
t 2.258 3.735 1.991 2.258 3.665 1.991 2.258 3.665 1.959 1.000 -0.000 0.000
t 2.258 3.735 1.991 2.258 3.665 1.959 2.258 3.735 1.959 1.000 -0.000 0.000
```

La siguiente etapa en el proceso es el **trazado de rayos de iluminación directa**. En esta etapa se toma como entrada la descripción de geometría producida en la etapa anterior sobre la que se realiza un proceso de trazado de rayos simplificado. El objetivo de esta etapa es obtener un conjunto de puntos de intersección entre los rayos lanzados desde las fuentes de iluminación y los objetos definidos en la escena. De este modo, el resultado producido es un conjunto de puntos que corresponden a los impactos de iluminación directa sobre la escena.

Como se ha mencionado anteriormente, se distinguen entre dos tipos de mapas de iluminación directa: aquel producido por fuentes de iluminación situadas en el interior de la escena (internas) y el producido por fuentes situadas en el exterior de la escena (externas). Para especificar el carácter de cada fuente, el nombre de estas debe estar precedido por la cadena "*int*" o "*ext*" según sean internas o externas respectivamente. Cualquier fuente cuyo nombre no esté precedido por alguno de estos prefijos será ignorada por el trazador de impactos.

El trazador de impactos desarrollado en esta etapa hace un trazado de rayos estocástico desde cada fuente de iluminación. Algunos de los rayos trazados intersecarán con los polígonos definidos en la escena. Tales puntos de intersección corresponderán al conjunto de impactos producido como resultado en esta etapa.

Se distinguen dos tipos de fuentes de iluminación: lámparas y fuentes focales. La diferencia clave entre ambas es la forma de distribución de los rayos de luz. En las fuentes de tipo lámpara no se tiene en cuenta ningún vector de dirección de la luz. Como consecuencia, los rayos de luz trazados desde este tipo de fuente pueden tomar una dirección cualquiera dentro de la esfera con centro en el punto de localización de la fuente y de radio la unidad. Por el contrario, las fuentes de tipo focal tienen definido un vector de proyección de la luz (vector



(a) Fuente de tipo lámpara (b) Fuente de tipo focal

Figura 4.14: Tipos de fuentes de iluminación

normal). En este último tipo, los rayos trazados pueden tomar cualquier vector de dirección que cumpla la expresión (4.15).

$$\vec{d} \cdot \vec{N} = \left|\vec{d}\right| \cdot \left|\vec{N}\right| \cdot \cos\theta \geq 0 \tag{4.15}$$

donde $\vec{d}$ es el vector de dirección de un rayo cualquiera, $\vec{N}$ es el vector normal de la fuente focal y θ es el ángulo que forman $\vec{d}$ y $\vec{N}$.

Es decir, las fuentes de tipo focal trazarán sólo aquellos rayos en los que el producto escalar de su vector de dirección con el vector normal de la fuente resulte en un valor positivo o igual a cero. En la figura 4.14 puede verse un esquema del modo de distribución de rayos de ambos tipos de fuentes.

Este proceso de trazado de rayos debe lanzar de forma estocástica un alto número de rayos desde cada fuente de iluminación definida. En su versión más simple, para cada rayo trazado debe hacerse una comprobación de intersección de tal rayo con cada polígono definido en la escena, lo cual es una operación no trivial que envuelve varias multiplicaciones y divisiones. Esto es un problema en escenas con un número de polígonos relativamente alto (alrededor de los 20.000 polígonos) y varias fuentes de iluminación definidas. La complejidad en este caso está definida por la expresión (4.16).

$$N_f \cdot N_r \cdot N_p \tag{4.16}$$

donde N_f es el número de fuentes de iluminación definidas en el modelo, N_r es el número de rayos de luz lanzados por cada una de ellas y N_p es el número de polígonos definidos en el modelo.

Para solventar este problema se ha implementado e integrado una estructura de datos de ordenamiento espacial de polígonos con el fin de acelerar la búsqueda de estos (ver sección 3.2.3). La estructura de datos utilizada ha sido de tipo **jerarquía de volúmenes límite** (figura 4.15) [16].

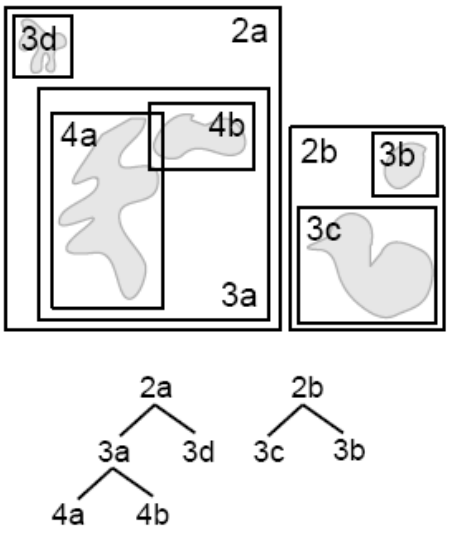


Figura 4.15: Esquema de jerarquía de volúmenes límite (BVH)

Las jerarquías de volúmenes límite tratan de organizar los polígonos definidos en un modelo 3D de acuerdo a su localización espacial. La organización realizada es en forma de árbol, donde cada nodo interior es un volumen límite que contiene todos los polígonos de los nodos hijos y cada nodo hoja contiene un polígono de la escena. Una vez el árbol está construido, se calcula la intersección de un rayo cualquiera con los volúmenes definidos por los nodos de primer nivel. Los nodos hijos de aquellos nodos con los que no se produzca una intersección serán automáticamente ignorados, mientras que los nodos hijos de aquellos nodos con los que sí existe intersección serán estudiados. Cuando el árbol está construido de acuerdo a una

buena heurística (una que de lugar a un árbol bien balanceado), esta técnica proporciona una reducción del número de comprobaciones muy interesante. La complejidad del algoritmo de trazado de rayos con esta estructura de datos incorporada es mejorada muy significativamente (ver expresión (4.17)).

$$N_f \cdot N_r \cdot \log_m N_p \tag{4.17}$$

donde N_f es el número de fuentes de iluminación, N_r es el número de rayos de luz trazados por cada una de ellas, m es el número medio de ramas por nodo en la estructura de árbol (en el caso de un árbol binario, $m = 2$) y N_p es el número de polígonos definidos en el modelo.

En los mapas de iluminación directa, el objetivo es también representar aquellos impactos de luz que provengan de fuentes con una alta energía luminosa con un color más cercano al blanco, mientras que aquellos que provengan de fuentes con baja energía luminosa deben ser representados con un color más cercano al negro. Para conseguir este propósito, cada impacto de la luz debe ser almacenado junto con un valor de color calculado de la siguiente forma. Ya que el resultado de este proceso son dos mapas de iluminación directa (uno para fuentes internas y otro para externas), todas las fuentes de la escena son clasificadas como *internas* o *externas*. Ahora, en cada clase, la fuente con mayor valor de energía luminosa asignará un valor de color máximo (es decir, blanco) a todos los impactos generados por rayos trazados por ella. El resto de fuentes asignarán un valor de color proporcional a su valor de energía luminosa a los impactos producidos como consecuencia de la intersección de los rayos trazados por ellas sobre los objetos de la escena. El resultado será por tanto un mapa donde los impactos con color más claros son los que han sido causados por fuentes de mayor energía luminosa y los impactos más oscuros, por fuentes con un menor valor de energía.

Por último, toda esta información es almacenada en un fichero de intercambio que servirá para que el conjunto de impactos generados sea importado al modelo de usuario con el objetivo de obtener una imagen 2D que represente el mapa de iluminación. A continuación puede verse un fragmento de un ejemplo de tal fichero de intercambio.

```
#################################################################
# YAReW Light Estimation Map Data File                          #
#                                                               #
# 00004-indoor.txt                                              #
```

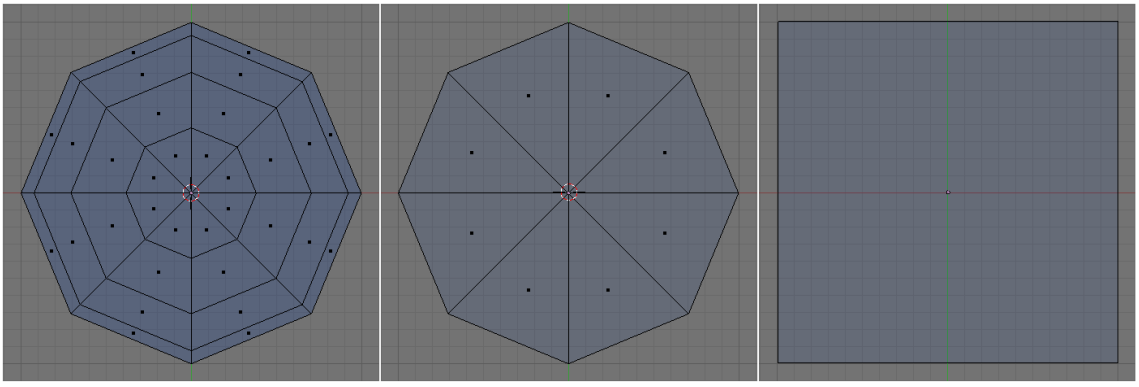
```
#####################################################################
# Col point (x, y, z) | White level [0.0, 1.0] |                   #

9.348245 5.312125 1.757572 1.000000
12.000000 11.669083 6.253784 1.000000
10.601804 8.808355 0.000000 1.000000
0.000000 16.883195 4.527619 1.000000
12.000000 6.454176 6.543398 1.000000
12.000000 9.108732 0.188063 1.000000
3.456477 18.000000 6.876735 1.000000
12.000000 10.618068 3.382792 1.000000
0.326806 18.000000 6.304043 1.000000
0.000000 17.040916 4.907825 1.000000
12.000000 7.888778 6.689961 1.000000
2.700068 4.830211 1.957902 1.000000
```

La siguiente etapa en el proceso es la **importación de impactos de iluminación directa** sobre el modelo proporcionado por el usuario. Esta etapa toma como entrada el fichero de impactos de iluminación directa generado en la etapa anterior y tiene como objetivo importar cada punto definido en tal fichero y representarlo como un impacto sobre la geometría de la escena de usuario.

El primer problema a resolver es responder a la pregunta de cómo representar cada punto de intersección rayo-polígono en la escena de usuario. El objetivo es conseguir la representación de cada punto de intersección mediante una primitiva gráfica que proporcione buena representación de impacto con la menor cantidad de recursos posible, ya que con el número de polígonos aumenta la cantidad de memoria y el tiempo de ejecución requeridos. En la versión del sistema presentada con este proyecto, tan sólo se incluye soporte para el paquete de software **Blender**. Así, se han estudiado las diferentes primitivas 3D proporcionadas en este software para la representación de los puntos de intersección identificados en la etapa anterior. En concreto, se ha prestado especial atención a la utilización de las primitivas *esfera*, *circunferencia* y *plano* (cuadrado). La figura 4.16 muestra estas primitivas representadas con Blender.

En primera instancia, se ha tratado de representar cada punto de intersección por medio de una **esfera**. Las ventajas de esta decisión son una representación natural del concepto de *impacto* entendido como un área con forma circular apreciada desde el punto de vista del observador con un color determinado. Además, la ventaja de utilizar esferas radica en la consecución del efecto de que independientemente del punto de vista del observador, este



(a) Esfera UV de resolución 8x8 (b) Circunferencia de resolución 8 (c) Plano

Figura 4.16: Algunas primitivas gráficas en Blender

observa el impacto como una circunferencia. La principal desventaja derivada de la utilización de esferas es la gran cantidad de polígonos que son necesarios para representar esta primitiva. Por ejemplo, la figura 4.16.(a) muestra una esfera UV con una resolución de 8 segmentos y 8 anillos. Esto hace un total de 64 polígonos necesarios para representar esta primitiva.

La siguiente primitiva estudiada ha sido la **circunferencia**. Al igual que ocurre en el caso de la esfera, la circunferencia es una buena forma de representación del concepto de impacto, aunque tiene la desventaja de perder la independencia con respecto a la localización y el ángulo de la cámara virtual para conseguir el efecto de visualizar una esfera desde esta. La solución a este problema se basa en colocar cada circunferencia en la escena de tal forma que el vector normal de cada cara sea de igual dirección que el vector de enfoque de la cámara, aunque de sentido contrario. Aún así, el número de polígonos necesarios para representar esta primitiva sigue siendo alto. Por ejemplo, en el caso de la figura 4.16.(b) se ha definido una circunferencia con una resolución de 8 segmentos, lo que hace necesarios 8 polígonos para su representación. Esto supone una gran mejora con respecto a los polígonos necesarios para representar una esfera, aunque sigue siendo un número elevado cuando se trata de representar varios miles de puntos de intersección.

Por último, se ha tenido en consideración la utilización de planos de forma cuadrada. Al contrario que ocurre con las dos primitivas anteriormente vistas, un plano cuadrado no es una forma muy natural de representar un impacto. Además, al ser un plano, tiene el mismo pro-

blema que las circunferencias, ya que es necesario establecer explícitamente una orientación del plano para que este sea perpendicular al vector de enfoque de la cámara. La aproximación seguida para solucionar este problema es el mismo que en el caso de las circunferencias: se orientan los planos de tal forma que su vector normal sea de igual dirección que el vector de enfoque de la cámara, pero de sentido contrario. La principal ventaja de utilizar esta primitiva es que sólo es necesario un polígono para su representación, por lo que existe un incremento lineal del número de polígonos necesarios con el número de puntos de intersección identificados. La figura 4.16.(c) muestra un ejemplo de esta primitiva en Blender.

Teniendo en cuenta las consideraciones anteriores, se ha optado por la utilización de planos para la representación de los puntos de intersección de los rayos de iluminación directa con los objetos de la escena. Esta decisión se ha tomado en favor de una máxima optimización del tiempo de ejecución requerido por el algoritmo importador a cambio de una representación menos natural.

Así, cada punto de intersección identificado en la etapa anterior es representado en la escena de usuario como un plano orientado de forma perpendicular al vector de enfoque de la cámara virtual y con un material con un valor de color en escala de grises proporcional al valor de energía luminosa especificado. De esta forma, aquellos impactos con un valor máximo tendrán colores cercanos al blanco, mientras que aquellos con un valor mínimo de energía tendrán colores cercanos al negro.

La última etapa en el proceso es la propia **generación de los mapas de iluminación directa**. Como se ha mencionado anteriormente, estos mapas son imágenes bidimensionales en blanco y negro en las que aparecen los impactos de los rayos de luz representados con un color dentro de la escala de grises.

En este punto, la escena de usuario está preparada con todos los puntos de intersección identificados en la etapa de trazado de rayos representados mediante primitivas gráficas. La forma más natural de generar los mapas de iluminación directa es *renderizando* la propia escena de usuario de tal forma que toda la escena aparezca de color negro, excepto las primitivas que se han añadido y que deben aparecer con una tonalidad diferente en escala de grises.

Para conseguir este propósito, cada material y textura de la escena de usuario es eliminado y sustituido por un material de color totalmente negro. De esta forma conseguimos el efecto

de *oscurecer* la escena de usuario. El material de cada primitiva de representación de los puntos de intersección fue establecido en la etapa anterior con un nivel de color en escala de grises determinado. En esta etapa, es necesario además establecer una opción *shadeless*[7] a estos materiales con el fin de evitar las sombras proyectadas por estas primitivas.

(a) Mapa de iluminación directa interna (b) Mapa de iluminación directa externa

Figura 4.17: Mapas de iluminación directa interna y externa

Con esta etapa, el proceso finaliza y da como resultado dos mapas de iluminación directa por cada escena de usuario: uno teniendo en cuenta el efecto de las fuentes de iluminación internas a la escena, y otro con las fuentes externas. En la figura 4.17 puede verse un ejemplo del resultado de este proceso.

Utilización de los mapas de iluminación directa

Una vez generados ambos mapas de iluminación directa para la escena de usuario, estos son utilizados para seleccionar la escena de la base de conocimiento más similar a la del usuario.

Cada escena en la base de conocimiento tiene asociada dos mapas de iluminación directa, por lo que la aproximación tomada para comparar dos escenas es medir la diferencia entre mapas de iluminación directa interna y externa. La medida tomada para medir esta diferencia es la raíz del error medio al cuadrado (*RMSD*) (ver sección 4.2.4), utilizada anteriormente en

[7]La razón de utilizar el término inglés es porque se hace referencia al valor de una opción concreta. La traducción de este término al castellano es *sin sombra*.

la generación de la base de conocimiento.

Para establecer el grado de similitud entre dos escenas (la proporcionada por el usuario y una dada de la base de conocimiento), se comparan sus respectivos mapas de iluminación directa interna y externa utilizando como medida la raíz del error medio al cuadrado. Los valores de diferencia entre ambos tipos de imágenes es sumada y este valor es el *grado de diferencia* entre ambas escenas. La escena de la base de conocimiento con un menor valor en esta medida es la escena seleccionada.

El último paso en esta etapa es seleccionar el conjunto de datos derivado de la escena de la base de conocimiento seleccionada para ser utilizado en el siguiente nivel de la arquitectura del sistema.

Extracción de parámetros de render

El otro objetivo perseguido en este nivel es la extracción de los parámetros que forman la configuración de render especificada por el usuario y su representación en un formato apropiado para ser procesado por el siguiente nivel de la arquitectura. Las figuras 4.5 y 4.6 muestran un ejemplo del resultado de esta etapa.

La forma de representación de la configuración de render del usuario es como tupla de pares atributo-valor. Esta forma es codificada de acuerdo al formato de fichero descrito en 4.2.5. Para el siguiente nivel, esta tupla de características es la instancia o ejemplar sobre el que se ha de estimar tanto el tiempo de render, como la calidad de resultado proporcionada. Además, es el punto de partida para las sugerencias de configuraciones optimizadas aportadas por el sistema.

4.3.3. Etapa de estimación y optimización

Este nivel en la arquitectura de YAReW comprende el llamado *núcleo de aprendizaje* del sistema. Aquí los objetivos son ofrecer estimaciones del tiempo de render y la calidad del resultado obtenido al renderizar el modelo del usuario de acuerdo a los valores de parámetros de render establecidos por él. Así mismo, la optimización de la configuración de parámetros establecida por el usuario es realizada en este nivel por un componente llamado *optimizador*

evolutivo. En este nivel tienen lugar todas las técnicas de Minería de Datos aplicadas sobre la base de conocimiento y la instancia de render del usuario.

Las entradas en este nivel son un ejemplar de render descrito mediante una tupla de pares atributo-valor que representa la configuración de parámetros de render establecida por el usuario y un conjunto de datos derivado de la escena de la base de datos identificada como la más similar a la escena de usuario. El resultado de la ejecución de los componentes de este nivel es una estimación del coste y rendimiento de una instancia del proceso de render lanzada sobre la escena de usuario y de acuerdo a la configuración de parámetros establecida por él. El coste del proceso corresponde al tiempo de ejecución necesario por la instancia del proceso de render lanzada, mientras que el rendimiento es definido como la calidad de la imagen producida. Además, en este nivel se proporcionan las sugerencias acerca de configuraciones de render óptimas según un criterio definido por el usuario y que será visto más adelante en esta sección. Una configuración óptima de parámetros de render es definida en el ámbito de este proyecto como aquella configuración de parámetros que conduce a maximizar la relación $\frac{\text{Calidad de Resultado}}{\text{Tiempo de Render}}$. La figura 4.18 muestra un esquema de la organización de los componentes en este nivel.

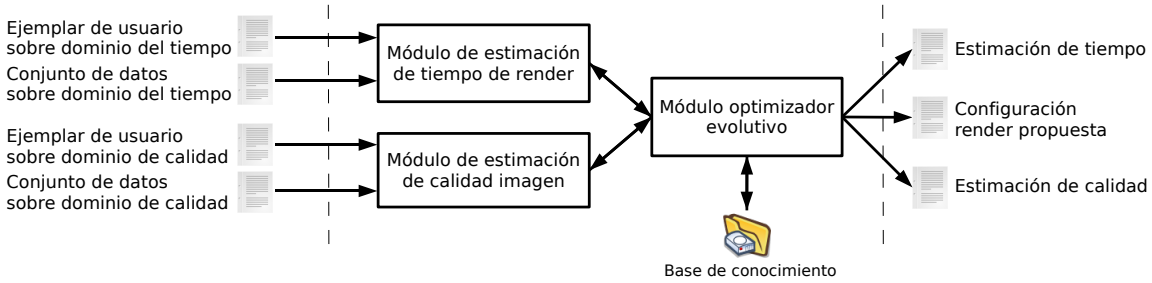


Figura 4.18: Componentes en el nivel de estimación y optimización

Los componentes incluidos en este nivel de la arquitectura son un **módulo de estimación de tiempo de render**, un **módulo de estimación de la calidad de imagen** y un **módulo de optimización evolutiva**. Los módulos de estimación de tiempo y calidad son envoltorios de funcionalidades ofrecidas por el paquete de software de minería de datos *Orange* [2]. El

optimizador evolutivo está basado en un algoritmo genético cuyo objetivo es la búsqueda de soluciones óptimas en un espacio de soluciones inicial llamado *población*. A continuación sigue una descripción en profundidad de cada uno de estos componentes.

Estimación de Tiempo de Render

Este componente tiene como objetivo realizar estimaciones del tiempo de ejecución sobre instancias de render proyectadas sobre el dominio del tiempo. Para cumplir su objetivo, hace uso de las funcionalidades proporcionadas por *Orange*, actuando en realidad como un envoltorio sobre este.

Ya que el tiempo de ejecución es una medida de carácter continuo, los algoritmos utilizados para realizar predicciones sobre esta característica deben ser apropiados para problemas de regresión. Los algoritmos utilizados en este proyecto para estimar el tiempo de render son **árboles de regresión** y ***k* vecinos más cercanos**. Una descripción en profundidad de las particularidades propias de estos algoritmos puede verse en los puntos 3.1.3 y 3.1.4, respectivamente, de este documento.

El algoritmo de inducción de árboles de regresión utilizado por este componente es una variante del algoritmo ID3 (ver 3.1.3), con algunas optimizaciones inspiradas en el algoritmo C4.5 desarrollado por Quinlan [26], como es el caso del manejo de atributos continuos. Ya que la estimación del tiempo constituye un problema de regresión, la medida utilizada a la hora de inducir el árbol en cuestión es el **error medio al cuadrado** (MSE). Así, en cada nodo intermedio se utilizará aquel atributo que provoque un menor error medio al cuadrado sobre los conjuntos en los que se dividen los datos.

Además de árboles de regresión, YAReW da la posibilidad al usuario de utilizar una técnica basada en estimación por vecindad como alternativa para predecir el tiempo de render de un ejemplar dado. Esta técnica está basada en el algoritmo de k vecinos más cercanos. El comportamiento exacto de este método puede ser especificado por medio de un conjunto de argumentos, como el número de vecinos a tener en cuenta (esta medida es la variable k), la medida de distancia utilizada y la posibilidad de ponderar las influencia de los vecinos identificados según la distancia de estos al ejemplar de test, o según posición en el ranking de distancia. Las medidas de distancia disponibles en la implementación utilizada son *Euclídea*

y *Manhattan*. Una descripción de las particularidades de todas estas puede verse en la sección 3.1.4 de este documento.

Estimación de Calidad de Resultados

Este es el componente encargado de proporcionar estimaciones sobre la calidad del resultado producido por una instancia de render determinada. Al igual que en el caso del componente de estimación de tiempo, este también actúa como una envoltura sobre las funcionalidades de Orange.

Al contrario que en el caso del tiempo de render, la calidad de imagen puede ser *etiquetada*, convirtiéndola así en un atributo nominal o categórico y, por tanto, de carácter discreto. Por este motivo, los algoritmos utilizados para realizar estimaciones sobre esta medida deben ser apropiados para resolver problemas de clasificación. Concretamente, las técnicas soportadas para *clasificar* la calidad de resultados son **árboles de clasificación**, ***k* vecinos más cercanos** y clasificador ***naive* bayes**. Puede verse una descripción en profundidad del comportamiento y las características propias de cada uno de estos métodos en las secciones 3.1.3, 3.1.4 y 3.1.5, respectivamente.

La implementación del algoritmo de inducción de árboles de clasificación utilizada por este componente es la misma que en el caso de la estimación del tiempo de render, con excepción de la medida utilizada para partir los conjuntos de ejemplares en cada nodo intermedio. En este caso, las medidas soportadas son **ganancia de información** y **grado de ganancia**[8], una medida similar a la ganancia de información donde se penalizan aquellos atributos que dan lugar a la división de un nodo interior en un número alto de nodos hijos.

La implementación del método de *k* vecinos más cercanos utilizada para clasificar la calidad de resultados es similar a la utilizada para predecir el tiempo de render, salvo que en este caso se incluye una medida especialmente útil en problemas de clasificación, además de las ya incluidas en la implementación del estimador de tiempo. Esta medida es denominada de *Hamming* (ver 3.1.4 para más detalles). Además, se mantienen el resto de opciones, como considerar la influencia de cada vecino identificado ponderando sus valores según posición en el ranking o según distancia con el ejemplar de test.

[8] Visto en la literatura como *gain ratio*

Además de los anteriores, este componente permite la utilización de un método de clasificación basado en aprendizaje bayesiano: el clasificador *naive* bayes. Este método es un referente en el campo del aprendizaje máquina, donde es muy utilizado como *benchmark* debido a que tiene un comportamiento muy eficiente en una gran variedad de problemas de clasificación diferentes. Este método trata de ofrecer una estimación de la probabilidad condicional (a posteriori) de la ocurrencia de un evento mediante la utilización de la probabilidad incondicional (a priori) y la verosimilitud de los datos manejados (ver 3.1.5). Además, como argumentos al algoritmo se permite la especificación del estimador de probabilidad utilizado. Las diferentes opciones son estimación mediante **frecuencias relativas** (ver expresión (4.18)), estimación de **Laplace** (ver expresión (4.19)) o estimación ***m*** (ver expresión (4.20)) [12].

Sea un conjunto de datos dado con N ejemplares clasificados en k clases diferentes y donde N_c ejemplares pertenecen a la clase c, se define la **frecuencia relativa** de ejemplares pertenecientes a la clase c como

$$\frac{N_c}{N} \tag{4.18}$$

Bajo el mismo supuesto, se define el estimador de probabilidad de **Laplace** como

$$\frac{N_c + 1}{N + k} \tag{4.19}$$

y, suponiendo que existe una probabilidad incondicional (a priori) de que un ejemplar determinado pertenezca a la clase c, se define el estimador de probabilidad ***m*** como

$$\frac{m \cdot P_c + N_c}{N + m} \tag{4.20}$$

Optimización de Configuraciones

El objetivo de este componente es proporcionar sugerencias de configuraciones de parámetros de render que maximicen la relación $\frac{\text{Calidad de Resultado}}{\text{Tiempo de Render}}$.

El problema de sugerir configuraciones óptimas de parámetros de render al usuario puede ser visto como un problema de búsqueda en un espacio de soluciones. Cada posible configuración de render es una solución. La calidad de las soluciones viene dada por una *función*

de idoneidad, la cual establece valores más altos a las soluciones con una mayor calidad y viceversa. Por tanto, el problema consiste en ir navegando a través del espacio de posibles configuraciones de render en busca de la solución óptima.

Se ha implementado un algoritmo genético para llevar a cabo tal búsqueda en el espacio de todas las posibles configuraciones de render. Los algoritmos genéticos basan su funcionamiento en el razonamiento de la teoría de la evolución propuesta por Darwin, la cual establece (a muy grandes rasgos) que la *calidad* de un individuo es propagada mediante la reproducción con otro individuo hasta sus descendientes. Así, los individuos descendientes de dos individuos con alta calidad, tendrán a su vez un valor alto de calidad.

Los algoritmos genéticos parten de un conjunto de soluciones inicial llamado **población**. Cada miembro de la población es evaluado según una **función de idoneidad** que define el criterio de calidad de los individuos. La transición hacia nuevas soluciones no conocidas se realiza mediante la operación de **cruce** entre dos soluciones conocidas. Esta operación consiste en combinar dos soluciones para dar lugar a una nueva solución. La hipótesis sobre la que se basa la operación de cruce es precisamente que los individuos descendientes de individuos de alta calidad tendrán también un valor alto de calidad.

Como se puede deducir de su funcionamiento, este tipo de algoritmos centran la búsqueda en aquellas zonas donde se identifican soluciones de alta calidad. Esto supone una gran eficiencia en la búsqueda, aunque a cambio de la obtención de máximos locales. Para aliviar en cierto grado el problema de localidad, se añade en el diseño del algoritmo una operación de **mutación**. Esta operación consiste en modificar algún aspecto de un individuo generado de forma aleatoria según una pequeña probabilidad. Esto provoca que la búsqueda *salte* a nuevas zonas del espacio de soluciones que serán estudiadas si los individuos identificados allí tienen valores de calidad altos o desestimadas en caso contrario.

La búsqueda finaliza cuando al cabo de un número determinado de iteraciones o cuando se ha alcanzado un valor de calidad especificado por el usuario.

En concreto, la implementación utilizada establece como función de idoneidad que el valor de calidad de un individuo es directamente proporcional a la relación $\frac{\text{Calidad de Resultado}}{\text{Tiempo de Render}}$. La población inicial sobre la que parte el algoritmo está formada por individuos tomados de dos grupos diferentes de soluciones: el compuesto por aquellas configuraciones de render

más similares a la de usuario y de mayor calidad y el compuesto por aquellas configuraciones óptimas conocidas. La proporción de individuos de uno y otro grupo en la población inicial es especificada mediante un valor de *afinidad* por el usuario.

El criterio de *afinidad con el usuario* es una medida que establece el grado de similitud de las configuraciones sugeridas por el sistema con la configuración de usuario. Un valor de afinidad máximo especifica al sistema que busque la configuración óptima en el conjunto de configuraciones más similares a la proporcionada por el usuario. Por el contrario, un valor de afinidad mínimo especifica al sistema que puede obviar la configuración de usuario y proporcionar la solución más óptima posible. Concretamente, el criterio de afinidad es establecido mediante un valor real en el intervalo $[0, 1]$ que especifica la proporción de individuos de ambos grupos en la población inicial.

A partir de la población inicial, el algoritmo escoge dos individuos de forma aleatoria y los somete a la operación de cruce. El resultado de esta operación es una nueva configuración de render desconocida, cuya calidad es evaluada según la función de idoneidad. Debido a que la nueva configuración ha sido generada por el algoritmo de búsqueda, los valores de tiempo de render y calidad de resultado de la misma son desconocidos. Es necesario, por tanto, estimar el tiempo de render y la calidad de resultado con la ayuda de los componentes anteriormente escritos en esta sección para poder evaluar la calidad de la nueva configuración según la función de idoneidad definida. Si la nueva configuración tiene un valor de idoneidad mayor que el de alguno de los individuos de la población, la nueva configuración es añadida a la población. La población es en realidad una lista de tamaño L especificado por el usuario que mantiene siempre los L elementos con mayor valor de idoneidad.

Cada configuración nueva generada mediante la operación de cruce es sometida con una pequeña probabilidad configurable por el usuario a la operación de mutación. Esta operación, cuando sucede, escoge un atributo al azar de la configuración generada y modifica su valor de acuerdo al rango de valores permitidos del atributo seleccionado. De este modo se trata de identificar configuraciones con un alto valor de idoneidad en zonas próximas a la zona de búsqueda del algoritmo.

Por último, el algoritmo finaliza al cabo de un número de iteraciones determinado por el usuario. El tiempo de ejecución del algoritmo, así como el nivel de optimización conseguido,

son directamente proporcionales al número de iteraciones ejecutadas.

4.3.4. Etapa de Presentación de Resultados

Este nivel comprende la interfaz de usuario del sistema. El objetivo de esta es proporcionar una forma sencilla de interacción con el sistema y presentar los resultados y toda la información relevante para el análisis de estos al usuario de una forma clara y atractiva.

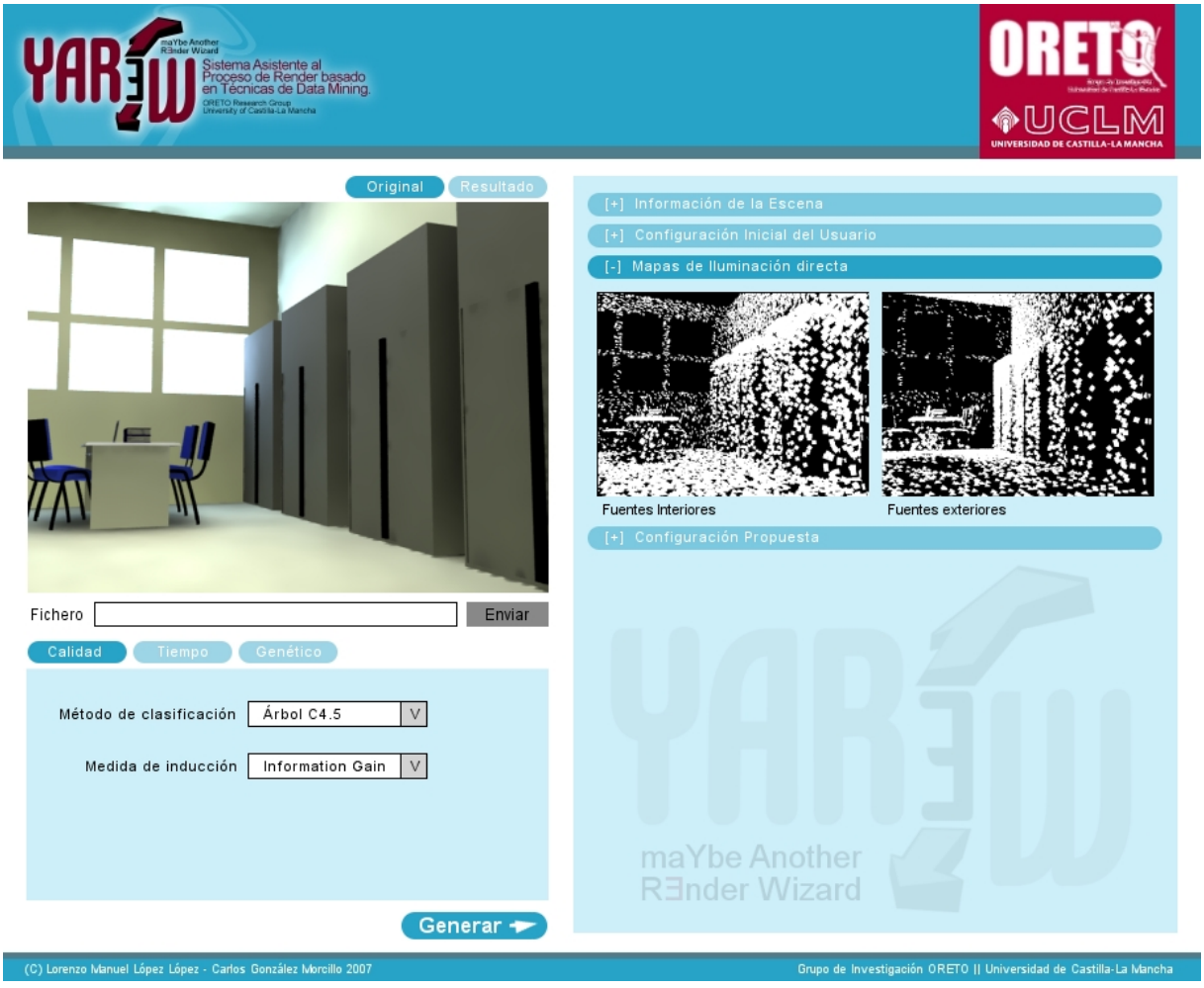


Figura 4.19: Interfaz de usuario del sistema

La interfaz de usuario desarrollada está basada en tecnologías web. En concreto se han utilizado el lenguaje de programación de aplicaciones web PHP [3] y tecnología basada en *Javascript Asíncrono con XML* (AJAX).

En la figura 4.19 se muestra una captura de pantalla de la interfaz de usuario del sistema.

Figura 4.20: Zonas en la interfaz de usuario del sistema

Según puede verse en la figura 4.20, la interfaz de usuario del sistema está dividida en tres zonas:

1. Zona de Render
2. Zona de Configuración
3. Zona de Resultados

La zona de render está compuesta por dos solapas: *original* y *resultado*. El usuario puede proporcionar una nueva escena al sistema mediante el control "*Enviar*" situado en la zona

inferior de la zona de render. Hecho esto, el sistema comienza a analizar la escena y realiza un render a baja resolución. El resultado de este proceso es la imagen que aparece en la zona de render bajo la solapa *original*. La solapa *resultado* muestra la imagen generada a partir del render de la escena de usuario con la configuración sugerida por el sistema.

La zona de configuración está compuesta por tres solapas diferentes: *Calidad*, *Tiempo* y *Genético*. Bajo la solapa *Calidad* se encuentran los controles que permiten al usuario configurar los métodos de clasificación empleados para estimar la calidad de la escena de usuario. Del mismo modo, bajo la solapa *Tiempo* se encuentran los controles que permiten al usuario especificar el método de estimación de tiempo y los parámetros que definen su comportamiento exacto. Por último, bajo la solapa *Genético* se encuentran aquellos controles que permiten la configuración del algoritmo de búsqueda genético encargado de proporcionar optimizaciones y sugerencias de configuraciones render. Una descripción en profundidad de todas las posibles opciones de configuración del comportamiento de los métodos de estimación de calidad y tiempo y del comportamiento del algoritmo de búsqueda genético puede verse en la sección 4.3.3 de este documento.

Por último, la zona de resultados está compuesta por cuatro paneles desplegables que agrupan cada uno un aspecto de los resultados producidos por el sistema. El panel *Información de la Escena* muestra algunas estadísticas de la escena de usuario, como número de polígonos o fuentes de iluminación. El panel *Configuración inicial de usuario* muestra los valores de los parámetros de render extraídos de la escena proporcionada por el usuario. El panel *Mapas de iluminación directa* muestra los mapas de iluminación directa interna y externa extraídos de la escena de usuario. El último panel, llamado *Configuración propuesta*, muestra la configuración sugerida por el sistema como resultado del análisis de la escena de usuario.

Capítulo 5

Resultados

5.1. Introducción

Este capítulo muestra un resumen de los resultados obtenidos tras ejecutar la fase de pruebas. Las pruebas realizadas han tenido el objetivo de demostrar la eficiencia del sistema en los siguientes términos.

- Capacidad de sugerir configuraciones de parámetros de render que maximicen la calidad de los resultados obtenidos y minimicen el tiempo de ejecución de los algoritmos de render empleados.
- Capacidad de ofrecer estimaciones precisas del tiempo de render requerido y la calidad del resultado producido por una instancia del proceso de render configurada según el usuario sobre una escena determinada.

Para la ejecución de la fase de pruebas, se ha creado una escena de interior con el generador de escenas desarrollado (ver sección 4.2). La figura 5.1.(a) muestra la configuración de parámetros de render establecida para la realización de la fase de pruebas. La figura 5.1.(b)

Raydepth	OSA	GIQuality	Depth	CDepth	PhCount	PhRadius	PhMixCount	ShQuality	Prec	Refine	EPower	GIPower
5	16	HIGH	3	1	100000	1.0	100	0.9	10	1.0	1.0	1.0

(a) Configuración de usuario establecida para las pruebas

(b) Imagen producida a raíz de la realización del render con la configuración expuesta arriba

Figura 5.1: **Arriba**: Configuración de parámetros render establecida para la fase de pruebas. **Abajo**: Resultado de realizar el render de la escena de pruebas con la configuración de parámetros indicada en (a) (ver versión a color en C.5)

muestra la imagen resultado de realizar el render de la escena de prueba con la configuración especificada en 5.1.(a).

La fase de pruebas ha sido diseñada para dar respuesta a las dos cuestiones apuntadas anteriormente.

En un primer paso, se pretende demostrar la eficiencia del sistema a la hora de ofrecer configuraciones de parámetros de render que maximicen la relación $\frac{\text{Calidad de Resultado}}{\text{Tiempo de Render}}$. Para ello,

se ha enviado al sistema la escena de prueba con la configuración de variables de render mostradas en la figura 5.1 y se han solicitado sugerencias según dos valores del atributo *afinidad con el usuario* (ver sección 4.3.3). Estos valores son 0.5 y 0.0. El primero indica al sistema que tiene cierta libertad a la hora de escoger la mejor configuración, pero que también debe tener en cuenta la configuración proporcionada por el usuario. El segundo, sin embargo, indica al sistema que tiene plena libertad a la hora de escoger la mejor configuración de render que encuentre, ignorando por completo si es necesario la configuración proporcionada por el usuario.

En la segunda fase se pretende demostrar la capacidad del sistema a la hora de ofrecer estimaciones del tiempo de render y la calidad de los resultados obtenidos según las configuraciones de variables proporcionadas en la etapa anterior. Para ello, se realizará el render de la escena de prueba con cada configuración sugerida y se contrastarán las diferencias entre las estimaciones realizadas y los valores efectivos de ambas medidas.

Además, el algoritmo genético de búsqueda de configuraciones tiene cuatro niveles de optimización posibles, de los cuales depende el número de iteraciones realizadas durante el proceso de búsqueda. Estos niveles son *bajo*, *medio*, *alto* y *muy alto*.

5.2. Sugerencia de configuraciones de render

En esta etapa se ha enviado al sistema la escena de prueba generada con una configuración de variables de render determinada (ver figura 5.1) y se han solicitado sugerencias de configuraciones según dos valores diferentes de la variable de *afinidad con el usuario*. Estos valores han sido 0.5 y 0.0. El primer valor indica al sistema que dispone de cierta libertad a la hora de escoger entre las mejores configuraciones encontradas, pero que también debe escoger alguna que se asimile en cierto grado a la proporcionada por el usuario. El segundo valor indica al sistema que dispone de cierta libertad a la hora de escoger la mejor configuración y que puede ignorar la proporcionada por el usuario si esta no tiene un buen valor de idoneidad.

En la figura 5.2.(e) pueden verse las configuraciones sugeridas por el sistema habiendo especificado un grado de afinidad 0.5 y según los diferentes niveles de optimización del algoritmo de búsqueda. Las figuras 5.2.(a) a 5.2.(d) muestran los resultados de realizar el render

(a) Resultado con un nivel de optimización *bajo*

(b) Resultado con un nivel de optimización *medio*

(c) Resultado con un nivel de optimización *alto*

(d) Resultado con un nivel de optimización *muy alto*

Nivel de Optimización	Raydepth	OSA	GIQuality	Depth	CDepth	PhCount	PhRadius	PhMixCount	ShQuality	Prec	Refine	EPower	GIPower
Bajo	3	16	HIGH	1	5	27485	10.9	67	0.173	38	0.662	12.5	0.14
Medio	32	11	BEST	1	5	100000	1.0	100	0.173	38	0.662	12.5	0.14
Alto	23	5	HIGH	1	5	27485	2.62	146	0.173	38	0.662	20.5	0.14
Muy Alto	26	11	BEST	1	5	27485	13.02	54	0.151	39	0.467	1.9	5.28

(e) Configuraciones de parámetros render sugeridas según el nivel de optimización empleado

Figura 5.2: Imágenes producidas al realizar el render de la escena de prueba con las distintas configuraciones sugeridas por el sistema con un valor de afinidad 0.5 (ver versión a color en C.6)

de la escena de prueba con cada configuración sugerida. Como puede observarse, las configuraciones sugeridas tienen cierta similitud con la configuración establecida inicialmente por el usuario (figura 5.1.(a)), aunque este no sea demasiado elevado. Este hecho puede deber-

se a que la configuración inicial establecida no es una configuración de buena calidad. Así, aprovechando el grado de libertad que hemos indicado al sistema, este ha escogido aquellas configuraciones con mayor valor de idoneidad de entre un conjunto compuesto por las mejores de las más similares a la configuración de usuario y las mejores absolutas contenidas en la base de conocimiento. La suposición de que la configuración inicial no tiene un gran valor de idoneidad viene reforzada además por el hecho de que la imagen resultado de realizar el render con dicha configuración produce un resultado claramente mejorable. A su vez, como puede observarse en las imágenes resultado en la figura 5.2, no es hasta el resultado del nivel de optimización más alto que se produce una mejora substancial sobre el resultado de la configuración inicial.

La figura 5.3.(e) muestra el conjunto de configuraciones de render sugeridas tras haber especificado un grado de afinidad 0.0 y según los diferentes niveles de optimización de la búsqueda. Las figuras 5.3.(a) a 5.3.(d) muestran las imágenes producidas tras realizar el render de la escena de prueba con las configuraciones sugeridas. En este caso, podemos observar que, en general, las configuraciones sugeridas tienen un grado de similitud con la configuración inicial menor. Esto se debe a que esta vez se ha indicado al sistema que dispone de plena libertad a la hora de elegir las configuraciones ofrecidas y refuerza aún más la hipótesis apuntada anteriormente de que la configuración inicial no es de muy buena calidad. Como puede verse en las imágenes resultado, esta vez sí que se han producido mejoras substanciales con respecto al resultado de la configuración inicial.

En ambos casos, es interesante observar cómo a través de cada nivel de optimización ciertos fragmentos de las configuraciones propuestas son mantenidas, mientras que otras son modificadas levemente. Esto se debe a que hay ciertos fragmentos que están presente normalmente en las configuraciones con mayor valor de idoneidad, con lo que son mantenidos a lo largo de todo el proceso. Los fragmentos restantes son modificados levemente en cada iteración con el objetivo de formar configuraciones completas con un buen valor de idoneidad.

(a) Resultado con un nivel de optimización *bajo*

(b) Resultado con un nivel de optimización *medio*

(c) Resultado con un nivel de optimización *alto*

(d) Resultado con un nivel de optimización *muy alto*

Nivel de Optimización	Raydepth	OSA	GIQuality	Depth	CDepth	PhCount	PhRadius	PhMixCount	ShQuality	Prec	Refine	EPower	GIPower
Bajo	36	8	BEST	1	5	273863	33.7	454	0.312	39	0.564	0.23	13.54
Medio	36	5	LOW	1	6	100927	1.9	193	0.185	44	0.752	0.58	11.13
Alto	36	8	BEST	6	1	96634	1.9	384	0.185	38	0.752	0.58	11.13
Muy Alto	28	11	BEST	1	5	96634	1.9	193	0.185	44	0.214	0.58	11.13

(e) Configuraciones de parámetros render sugeridas según el nivel de optimización empleado

Figura 5.3: Imágenes producidas al realizar el render de la escena de prueba con las distintas configuraciones sugeridas por el sistema con un valor de afinidad 0.0 (ver versión a color en C.7)

5.3. Estimación de tiempo de render y calidad de resultados

Esta segunda etapa de la fase de pruebas tiene como objetivo demostrar la capacidad del sistema para realizar estimaciones precisas del tiempo de render y la calidad de los resulta-

dos según las configuraciones de render sugeridas por el sistema en la sección anterior. Así, esta sección ofrece un contraste entre los valores estimados y efectivos para ambas medidas: tiempo de render y calidad de resultados. Las tablas 5.1 y 5.2 muestran la relación entre los valores estimados y reales de ambas medidas.

Nivel de Optimización	**Tiempo de Render**		**Calidad de Resultado**	
	Estimado	**Real**	**Estimado**	**Real**
Bajo	62	30	8	8
Medio	119	59	8	8
Alto	62	35	8	8
Muy Alto	63	66	9	9

Cuadro 5.1: Resultados de la realización de las pruebas con un valor de afinidad 0.5

Nivel de Optimización	**Tiempo de Render**		**Calidad de Resultado**	
	Estimado	**Real**	**Estimado**	**Real**
Bajo	114	165	8	8
Medio	101	42	8	9
Alto	119	87	9	9
Muy Alto	114	75	9	9

Cuadro 5.2: Resultados de la realización de las pruebas con un valor de afinidad 0.0

Tiempo (error medio cometido)	**Calidad (porcentaje de aciertos)**
37.875 segundos	87.5 %

Cuadro 5.3: Rendimiento del sistema en la realización de las pruebas

Tras realizar el render de la escena de prueba con cada configuración sugerida en la etapa anterior, los tiempos de render requeridos han sido anotados con el objetivo de observar el error cometido en las predicciones. Como puede observarse en la tabla 5.3, el error medio cometido en las estimaciones del tiempo de render es de 37.875 segundos. Este resultado

puede ser debido al efecto que tiene sobre el tiempo de render los atributos relativos a la naturaleza de la escena, como número de polígonos, número de fuentes de ilumianación, materiales, texturas o *shaders*. Como se ha comentado anteriormente en este documento, no ha formado parte de los objetivos de este proyecto modificar la estructura interna de las escenas estudiadas, por lo que una posible solución a este problema sería refinar el proceso de emparejamiento de la escena de usuario con la escena más similar contenida en la base de conocimiento.

En cuanto a la calidad de los resultados, una vez realizado el render de la escena de prueba con cada configuración sugerida en la etapa anterior, se ha consultado a un experto para que valore la calidad de las imágenes producidas con respecto a la calidad del resultado inicial de la escena de prueba. Las tablas 5.1 y 5.2 muestran la calidad estimada y la etiquetada por el experto en cada caso. Según puede verse en la tabla 5.3, el error cometido en cuanto a la predicción de la calidad de una escena ha sido del 12.5 % del total de configuraciones analizadas.

Capítulo 6

Conclusiones y Propuestas

6.1. Conclusiones

En este capítulo dedicado a las conclusiones, se comenta en qué medida ha sido logrado cada uno de los objetivos propuestos.

- **Proporcionar sugerencias de configuraciones de parámetros de render que maximicen la calidad de los resultados obtenidos y minimicen el tiempo de ejecución de los algoritmos de render utilizados**: Se ha demostrado la capacidad del sistema para proporcionar sugerencias de configuraciones de render que maximizan la calidad de los resultados producidos y minimizan el tiempo de cómputo requerido por los métodos de render (ver sección 5.2). La calidad de las sugerencias ofrecidas por el sistema es, normalmente, directamente proporcional al nivel de optimización especificado para el algoritmo genético de búsqueda.

- **Estimación del tiempo de render y la calidad del resultado prevista para una escena y una configuración de parámetros de render proporcionados por el usuario**:

Se ha demostrado la capacidad del sistema para proporcionar estimaciones del tiempo de render y la calidad de los resultados producidos según diferentes configuraciones de variables. Aunque el error medio cometido en la estimación del tiempo de render es relativamente alto, la precisión en las estimaciones realizadas permiten es suficientemente buena como para ofrecer sugerencias de configuraciones render que maximizan la relación de idoneidad $\frac{\text{Calidad de Resultado}}{\text{Tiempo de Render}}$. Por otro lado, la precisión en las estimaciones de calidad de resultados obtenida durante las pruebas demuestra la capacidad del sistema para ofrecer estimaciones relativamente fiables de la calidad de los resultados que serán producidos por una configuración de render determinada sobre una escena dada.

El caso de los sub-objetivos es comentado a continuación:

- **Construcción de una plataforma de análisis de datos soporte a la decisión genérica**:
  - *Genérico respecto a los métodos de render empleados*: El sistema es totalmente genérico con respecto a los posibles métodos de render utilizados. Esto se debe a que la funcionalidad y el rendimiento del sistema se basan en la base de conocimiento, la cual contiene un conjunto de datos obtenidos a partir de la experiencia. Cada instancia de render es almacenada como un dato en la base de conocimiento y es descrita mediante un conjunto de atributos que son dependientes de cada implementación de método de render utilizada. La base de conocimiento puede contener conjuntos de datos de diferentes métodos de render e instancias de render descritas mediante conjuntos de atributos diferentes. Las únicas limitaciones son que todas las instancias en un mismo conjunto de datos deben estar descritas con el mismo conjunto de atributos y que para estimar una instancia determinada descrita mediante un conjunto de atributos, se debe utilizar un conjunto de datos cuyo dominio esté compuesto por el mismo conjunto de atributos.
  - *Genérico respecto a los métodos de aprendizaje empleados*: Toda la información en la base de conocimiento y las instancias de render a estimar son representadas en ficheros de texto plano en formato tabular (ver sección 4.2.5). Este formato es uno de los más utilizados dentro del campo del aprendizaje y la minería de datos,

por lo que ha llegado a ser un *estándar de facto*. Cualquier método de aprendizaje o minería de datos con capacidad para procesar tal formato de fichero puede ser integrado en el sistema sin problema alguno.

- *Genérico respecto a las variables de render empleadas*: Como se ha indicado anteriormente, las únicas limitaciones impuestas por el sistema son que todas las instancias contenidas en un mismo conjunto de datos deben estar representadas mediante el mismo conjunto de atributos y que para estimar una instancia determinada, esta y el conjunto de datos utilizado deben estar descritos mediante el mismo dominio. Con la excepción de esto, la base de conocimiento puede contener conjuntos de datos descritos mediante dominios diferentes y es posible agregar o eliminar atributos de un dominio siempre que se respeten las limitaciones apuntadas.

Estas propiedades hacen que el sistema desarrollado sea **adaptable** a nuevos métodos de render, software de usuario utilizado y métodos de aprendizaje o minería de datos empleados.

- **Instanciación de la plataforma creada para su utilización con el software empleado en el desarrollo de este proyecto**: Se ha desarrollado una instancia del modelo genérico del sistema para su integración con un software específico. Dicho software es el paquete de desarrollo de gráficos Blender [4] y el motor de render Yafray [6].

- **Desarrollar una plataforma robusta**: El sistema es capaz de ofrecer sugerencias y estimaciones para cualquier tipo de escena proporcionada por el usuario. No obstante, tales respuestas serán precisas o no dependiendo del grado de similitud de la escena proporcionada por el usuario con alguna de las contenidas en la base de conocimiento. Cuando una escena de un tipo determinado es incluido en la base de conocimiento, el sistema es dotado con la capacidad para ofrecer estimaciones y sugerencias con cierta precisión para cualquier escena similar.

- **Desarrollo de un sistema multi-plataforma**: El desarrollo del sistema se ha realizado utilizando tecnologías y lenguajes de programación que aseguran una portabilidad

instantánea a través de las diferentes plataformas.

- **Desarrollo del sistema conforme a estándares abiertos**: Se ha utilizado el formato de fichero tabular como medio de representación del conocimiento utilizado por los métodos de aprendizaje y minería de datos (ver sección 4.2.5).
- **Desarrollo del sistema utilizando software de código abierto**: Todo el software utilizado y producido durante la realización de este proyecto está publicado bajo alguna licencia de código abierto reconocido por la *Open Source Initiative* [1].

6.2. Líneas de investigación abiertas

YAReW es la primera propuesta de sistema que trata de dar asistencia al usuario en el proceso de render mediante técnicas de análisis de datos basadas en Data Mining. Por este motivo, son varios los aspectos del sistema que pueden ser investigados en más profundidad con el objetivo de ir puliendo detalles y poder así ofrecer mejores sugerencias y estimaciones para un conjunto potencialmente mayor de escenas de interior.

Las distintas propuestas aquí mencionadas son clasificadas en dos grupos atendiendo la parte del desarrollo que tratan de optimizar. Así, se identifican propuestas orientadas a optimizar la generación de contenidos de la base de conocimiento y orientadas a optimizar algún aspecto del sistema.

6.2.1. Propuestas para optimizar la generación de conocimiento

Los contenidos de la base de conocimiento tienen una gran influencia en el rendimiento del sistema. Por ello, es importante generar unos contenidos de alta calidad que permitan al sistema ofrecer sugerencias y estimaciones de la forma más precisa posible.

En el proceso de generación de la base de conocimiento, la etapa de *bootstrapping* es la encargada de generar un conjunto de configuraciones de render diferentes para una escena determinada (ver sección 4.2.2). Esta etapa tiene una importancia vital, ya que la muestra de datos sobre la que trabajarán los métodos de predicción y estimación es generada aquí.

Cada configuración diferente en esta etapa es generada completamente al azar. Es decir, a cada parámetro de render se le asigna un valor aleatorio dentro del rango de valores que este admite. Como se ha comentado anteriormente en la sección 4.2.2, esta aproximación tiene la ventaja de generar una muestra aleatoria que no está sesgada por ningún criterio.

Sin embargo, algunos parámetros de render tienen sentido en situaciones muy concretas. Por ejemplo, el parámetro que establece la profundidad de recursividad para efectos de cáusticas (ver sección 4.2.1) tiene sentido ser establecido sólo en escenas donde hayan sido definidos materiales translúcidos que den lugar a tales efectos.

Una posible optimización de esta etapa consiste en realizar un estudio previo de la geometría y las propiedades de la escena de usuario con el objetivo de identificar ciertas características en esta que permitan *acotar* el rango de valores que asignar a ciertos parámetros. De esta forma se reduciría el *ruido* en los datos generados y se daría lugar a conjuntos de datos de mayor calidad que permitirían una actuación más eficiente de las técnicas de aprendizaje utilizadas.

Otro aspecto que puede ser optimizado es el proceso de generación de los contenidos de la base de conocimiento. La base de conocimiento está compuesta por un conjunto de escenas de interior diferentes que cubren un espectro determinado. Cada una de estas escenas tiene asociado un conjunto de ejemplares de render que son utilizados por las técnicas de minería de datos para dar estimaciones sobre las escenas proporcionadas por el usuario. Es importante que estos conjuntos contengan un gran número de instancias de render, ya que la calidad de las hipótesis inducidas por los métodos de aprendizaje máquina viene dada en gran parte por el número de instancias de render analizadas, siendo normalmente el grado de aproximación de la hipótesis inducida con la hipótesis real directamente proporcional al número de ejemplares de entrenamiento estudiados. Así, el proceso de estimación actual consiste en, dada una escena proporcionada por el usuario, identificar la escena de la base de conocimiento que más se parezca a la de usuario y utilizar su conjunto de datos para ofrecer estimaciones de la forma más precisa posible.

La situación ideal sería aquella en la que la propia escena de usuario es sometida al proceso de bootstrapping y render de tal forma que quedaría incluida en la base de conocimiento. De este modo se permitiría utilizar el conjunto de ejemplares derivados de la misma escena

de usuario a la hora de ofrecer las estimaciones oportunas.

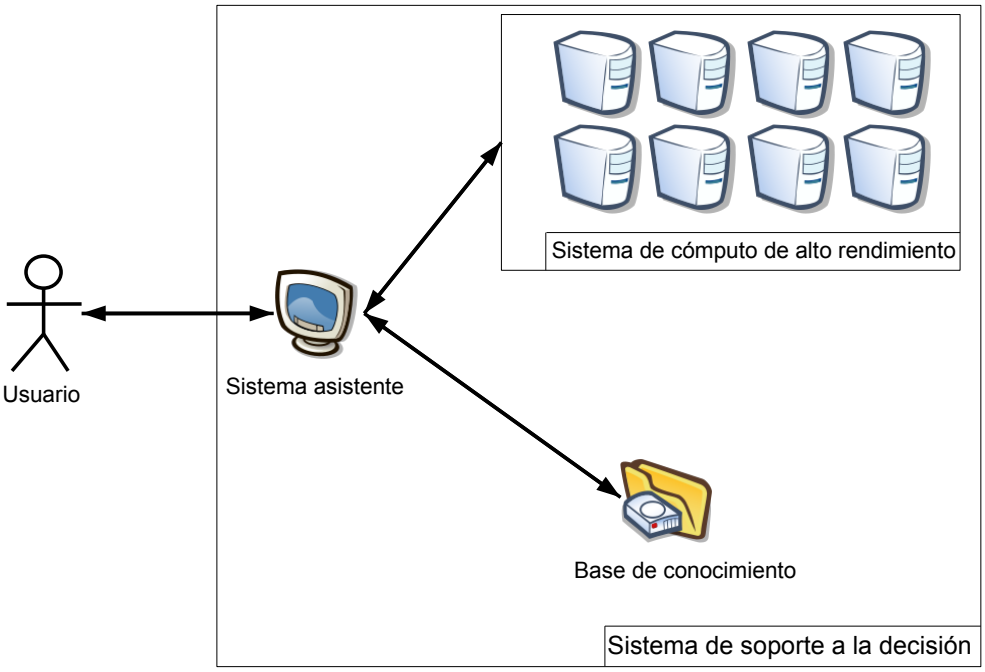


Figura 6.1: Propuesta de integración de un sistema de computación de alto rendimiento

El mayor problema para llevar a cabo esta situación es la cantidad de tiempo de cómputo requerido para ejecutar un conjunto relativamente grande de instancias de render para una escena dada. Este problema podría ser resuelto con la utilización de aproximaciones basadas en computación de alto rendimiento (figura 6.1). En concreto, podría utilizarse un grid computacional o un cluster de computadores situado detrás del sistema asistente con el fin de satisfacer sus necesidades de cómputo. Así se podría dar lugar a una situación en la que, a partir de una escena proporcionada por el usuario, el sistema podría someter dicha escena al proceso de bootstrapping y render con el objetivo de poder utilizar las propias instancias de render derivadas de la misma escena de usuario para ofrecer las estimaciones oportunas.

6.2.2. Propuestas de optimización del sistema asistente

La arquitectura del sistema asistente está dividida en tres niveles:

- Nivel de análisis y preproceso de escena.
- Nivel de estimación y optimización de configuraciones.
- Nivel de presentación de resultados.

A continuación se detallan algunas propuestas dirigidas a optimizar ciertas características de cada nivel.

Nivel de análisis y preproceso de escena

En este nivel tienen lugar todas aquellas operaciones dirigidas a identificar la escena de la base de conocimiento más similar a la escena de usuario. Actualmente, se realiza una comparación de escenas según la cantidad de luz que estas reciben desde las fuentes de iluminación definidas. Este procedimiento se realiza mediante la extracción y comparación de mapas de iluminación directa de fuentes internas y externas a la escena (ver sección 4.3.2).

Esta aproximación funciona muy bien en gran variedad de casos diferentes, ya que el tiempo de cómputo de los algoritmos de iluminación global está muy influenciado por la cantidad de luz que reciben los objetos definidos en la escena. Sin embargo, hay ciertas características que deberían tenerse también en cuenta a la hora de decidir qué escena de la base de conocimiento es más apropiada para ofrecer estimaciones sobre una escena de usuario. Algunas de estas características son el número y naturaleza de los materiales y texturas definidos en la escena, naturaleza de los caminos seguidos por la luz desde su salida del punto de vista del observador hasta una fuente de iluminación, número de objetos y polígonos definidos en la escena, etc.

Para tratar de refinar el proceso de identificación de la escena de la base de conocimiento más apropiada para una escena de usuario dada, se propone un proceso jerárquico en forma de árbol de decisión. En esta estructura, cada nodo intermedio corresponde a un test a realizar sobre la escena de usuario y cada nodo hoja corresponde a una escena de la base de conocimiento identificada como la *más apropiada*. Así, el proceso de emparejamiento comenzaría con la escena de usuario sometida al test especificado en el nodo raíz del árbol de decisión. Según la salida de dicho test, el proceso continuaría sometiendo la escena de usuario al test

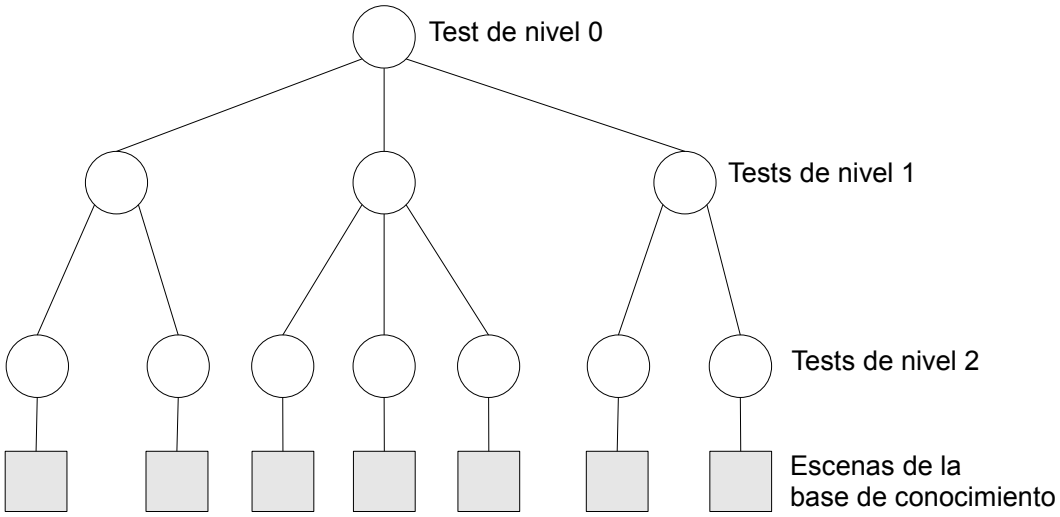


Figura 6.2: Estructura de decisión para el emparejamiento de escenas

especificado por el siguiente nodo intermedio. El proceso continúa así sucesivamente hasta que se llegue a un nodo hoja, el cual identifica la escena de la base de conocimiento más apropiada para realizar las estimaciones y sugerencias oportunas sobre la escena de usuario. La figura 6.2 muestra un esquema de la estructura de datos de decisión propuesta.

Ejemplos de algunos test relevantes para ser incluidos en los nodos intermedios de dicha estructura son:

- Relación de cantidad de luz absorbida por la escena y proveniente de fuentes de iluminación internas y externas.
- Número y naturaleza de materiales definidos en los objetos de la escena.
- Número y naturaleza de texturas definidas en los objetos de la escena.
- Número de objetos identificados en el plano de escena.
- Número de polígonos identificados en el plano de escena.
- Naturaleza de los caminos de luz identificados desde su salida en el punto del observador (cámara virtual) hasta su llegada a alguna fuente de iluminación.

Nivel de estimación y optimización de configuraciones

Este nivel comprende el núcleo de aprendizaje del sistema y, como tal, en él tienen cabida todos los métodos y técnicas de estimación derivadas del campo de minería de datos.

En la versión del sistema actual, se ha dado soporte para la especificación de ciertos parámetros propios de cada método que permiten especificar al usuario el comportamiento exacto deseado de cada algoritmo. Esto es especialmente apropiado para usuarios con un conocimiento superficial acerca del funcionamiento propio de las técnicas utilizadas. No obstante, sería interesante incluir ciertas características que dotasen de más flexibilidad de configuración a aquellos usuarios expertos que lo necesitasen.

Ejemplos de tales características deseables son:

- Posibilidad de hacer análisis de importancia de los atributos del dominio de datos. De esta forma se podría hacer un ordenamiento por influencia de los atributos del dominio sobre el modelo de datos para poder restringir la inducción de hipótesis a la utilización de un subconjunto de todos los atributos. Esto es de gran interés en la utilización de técnicas de predicción basadas en instancias, como el modelo de k vecinos más cercanos o técnicas de aprendizaje bayesiano.

- Posibilidad de realizar análisis estadísticos sobre los datos para identificar ejemplares extraordinarios (ruido) y permitir al usuario su eliminación del conjunto de datos de entrenamiento.

- Posibilidad de identificar atributos relacionados entre sí y sustituirlos por atributos agregados definidos por el usuario. En ciertos métodos, como aquellos basados en aprendizaje bayesiano, esta aproximación podría dotar al sistema de una mayor precisión en las predicciones.

- Posibilidad de realizar test estadísticos descriptivos para observar la distribución de los datos con respecto a atributos o magnitudes definidas por el usuario. Por ejemplo, se podría tratar de visualizar cómo influye un cierto atributo aislado en el tiempo de render o la calidad de los resultados producidos.

Nivel de presentación de resultados

En este nivel tienen cabida todas las técnicas de visualización de resultados. En la versión actual, el sistema muestra las estimaciones de tiempo de render y calidad de resultados que resultan de la ejecución de la etapa anterior.

Algunas propuestas de mejora que tienen cabida en este nivel son las siguientes.

- Inclusión de un módulo de evaluación mediante el cual el usuario tenga la posibilidad de observar qué eficiencia demuestra cada una de las técnicas de predicción utilizadas.
- Visualización de información relativa a las técnicas de predicción utilizadas. Por ejemplo, en el caso de árboles de decisión (clasificación o regresión), el usuario podría visualizar el árbol inducido a raíz de los datos, o en el caso de técnicas de predicción basadas en instancias, el usuario podría visualizar una gráfica de proyección con las instancias de entrenamiento y test representadas como puntos proyectados en ella.
- Visualización de patrones, en formas de reglas de decisión o de asociación, encontrados en los datos.
- Visualización de la influencia de ciertos parámetros de render sobre alguna característica del resultado final.

Mejoras con respecto a la interacción con el usuario

Se puede tratar la posibilidad de integrar el sistema con algún software de desarrollo de gráficos 3D, como Blender, para proporcionar al usuario una interacción con el sistema transparente y en tiempo real.

La idea es proporcionar al usuario información en tiempo real conforme él va modificando parámetros de render en Blender. Así, ante un cambio en la configuración realizado por el usuario, este podría visualizar inmediatamente las consecuencias de sus decisiones, aportando así una mayor rapidez y flexibilidad en el desarrollo. Así mismo, el sistema debería proporcionar sugerencias de configuraciones en tiempo real según los criterios de afinidad definidos por el usuario (ver sección 4.3.3).

Esta integración podría llevarse a cabo en forma de *plugin* para Blender. De esta forma, el usuario dispondría de una ventana de asistencia al usuario que en todo momento proporcionaría información en tiempo real y de forma transparente.

Apéndice A

Código Fuente

Debido a la extensión total del código fuente desarrollado durante el proyecto, en este anexo sólo se incluye el código relativo a algunos de los componentes más importantes del sistema desarrollado. La mayor parte del código desarrollado ha sido escrito en lenguaje Python, excepto en el caso del trazador de rayos de iluminación directa, el cual ha sido codificado en lenguaje C++, por razones de eficiencia. El código aquí listado corresponde a los siguientes componentes:

- **Bootstrapping**: Se muestra el código encargado de generar diferentes configuraciones de render aleatorias para una misma escena.
- **Evaluador de calidad de imagen**: Se ha incluido la implementación del método de proyección de histograma.
- **Trazador de rayos de iluminación directa**: Se incluye la clase *RayTracer*, que implementa la funcionalidad básica del trazador de rayos de iluminación directa. Este componente ha sido codificado en lenguaje C++.
- **Importador de impactos de iluminación directa**: Este programa tiene como objetivo representar los impactos de iluminación directa generados por el trazador de rayos en la escena de usuario y obtener el mapa de iluminación directa.
- **Optimizador genético de configuraciones render**: Código encargado de proporcionar configuraciones de render que maximicen la relación $\frac{\text{Calidad de Imagen}}{\text{Tiempo de Render}}$.

A.1. Bootstrapping

```
#!BPY

import Blender
from Blender import Scene, World
from Blender.Scene import Render

import random, os, os.path

DEFAULT_NSCENES = 1

GI_QUALITY = ['LOW', 'MEDIUM', 'HIGH', 'HIGHER', 'BEST']

FILENAME = Blender.Get('filename')

def saveNewFile(filename):
    Blender.Save(os.path.join(os.getcwd(), filename))

def Run(renderData, debug=True):
    random.seed(None)

    nTimes = os.getenv('bootstrap_nscenes', -1)
    if nTimes == -1:
        nTimes = DEFAULT_NSCENES
    else:
        nTimes = int(nTimes)

    for i in range(nTimes):
        filename = "%s-%d.blend" % (FILENAME[:-6], i)

        # Set some settings to predetermined
        renderData.setRenderer(Render.YAFRAY)
        renderData.enableShadow(1)
        renderData.enableRayTracing(1)
        renderData.enableEnvironmentMap(1)
        renderData.setRenderWinSize(100)
        renderData.sizePreset(Render.PC)
        renderData.imageType = Render.PNG
        renderData.enableRGBColor()

        # Render ray depth from camera
        raydepths = random.randint(1, 40)
        renderData.yafrayRayDepth(raydepths)

        # Oversampling level
        renderData.enableOversampling(1)
        osalevels = [5, 8, 11, 16]
        renderData.setOversamplingLevel(osalevels[random.randint(0,
                                                len(osalevels) - 1)])

        # Yafray GI Method
        renderData.setYafrayGIMethod(Render.GIFULL)
        # Enable Yafray GI cache
```

```
renderData.enableYafrayGICache(1)

# Yafray GI Quality
q = random.randint(0, len(GI_QUALITY) - 1)
if GI_QUALITY[q] == 'LOW':
    renderData.setYafrayGIQuality(Render.LOW)
elif GI_QUALITY[q] == 'MEDIUM':
    renderData.setYafrayGIQuality(Render.MEDIUM)
elif GI_QUALITY[q] == 'HIGH':
    renderData.setYafrayGIQuality(Render.HIGH)
elif GI_QUALITY[q] == 'HIGHER':
    renderData.setYafrayGIQuality(Render.HIGHER)
else: # GI_QUALITY[q] == 'BEST':
    renderData.setYafrayGIQuality(Render.BEST)

# Yafray GI Depth. Value must be between 1-8
depth = random.randint(1, 8)
renderData.yafrayGIDepth(depth)

# Yafray GI CDepth. Value must be between 1-8
cdepth = random.randint(1, 8)
renderData.yafrayGICDepth(cdepth)

# Enable photons
renderData.enableYafrayGIPhotons(1)

# Yafray GI Photon Count
nphotons = random.randint(10000, 500000)
renderData.yafrayGIPhotonCount(nphotons)

# Yafray GI Photon Mix Count
nmixphotons = random.randint(30, 500)
renderData.yafrayGIPhotonMixCount(nmixphotons)

# Yafray GI Photon Radius
pradius = random.uniform(0.5, 50.0)
renderData.yafrayGIPhotonRadius(pradius)

# Yafray GI Pixels per sample
pps = random.randint(1, 50)
renderData.yafrayGIPixelsPerSample(pps)

# Yafray GI Refinement
refine = random.uniform(0.1, 1.0)
renderData.yafrayGIRefinement(refine)

# Yafray GI Shadow Quality
sq = random.uniform(0.1, 1.0)
renderData.yafrayGIShadowQuality(sq)

# Yafray GI Power
power = random.uniform(0.1, 40.0)
renderData.yafrayGIPower(power)
```

```
        # Yafray GI Indirect lighting power.
        ipower = random.uniform(0.1, 40.0)
        renderData.yafrayGIIndirPower(ipower)

        # Save results into a new file
        filename = FILENAME[:-6] + "-%03d" % (i, ) + ".blend"
        saveNewFile(filename)

        ## DEBUGGING
        if debug:
            print filename
        ## END_DEBUGGING

Run(Scene.GetCurrent().getRenderingContext())
```

A.2. Proyección de histograma

```
#!/usr/bin/env python

import PIL.Image
import sys

def newImage(filename):
    return PIL.Image.open(filename)

def getImagePixels(image):
    if image.tell() != 0:
        image.seek(0)
    return image.load()

def getTotalPixels(hist):
    n = 0
    for i in hist:
        n += i

    return n

def histogramToProbDist(hist):
    pr = []
    npixels = getTotalPixels(hist)

    for i in range(len(hist)):
        pr.append(float(hist[i])/npixels)

    return pr

def getEqMatchingVector(img):
    eqhist = []
    hist = img.histogram()
    rhist = hist[:256]    # Histogram for R channel
    ghist = hist[256:512] # Histogram for G channel
```

```
    bhist = hist[512:768] # Histogram for B channel

    # Obtain probability distributions for each histogram
    rpr = histogramToProbDist(rhist)
    gpr = histogramToProbDist(ghist)
    bpr = histogramToProbDist(bhist)

    # Matching vector for R channel
    for i in range(len(rpr)):
        n = 0.0
        for j in range(i):
            n += rpr[j]
        eqhist.append(n + rpr[i])

    # Matching vector for G channel
    for i in range(len(gpr)):
        n = 0.0
        for j in range(i):
            n += gpr[j]
        eqhist.append(n + gpr[i])

    # Matching vector for B channel
    for i in range(len(bpr)):
        n = 0.0
        for j in range(i):
            n += bpr[j]
        eqhist.append(n + bpr[i])

    matvec = []
    for i in range(len(eqhist)):
        matvec.append(int(eqhist[i] * 255))

    return matvec

def HistogramMatchedImage(img_filename, masterimg_filename):
    # Image to match
    img = newImage(img_filename)
    eqmatvec = getEqMatchingVector(img)

    # Reference image
    masterimg = newImage(masterimg_filename)
    mastereqmatvec = getEqMatchingVector(masterimg)
    masterRChannel = mastereqmatvec[:256]
    masterGChannel = mastereqmatvec[256:512]
    masterBChannel = mastereqmatvec[512:768]

    # Start matching
    result = PIL.Image.new('RGB', img.size)

    imgPix = getImagePixels(img)
    resultPix = getImagePixels(result)

    for i in range(img.size[0]):
        for j in range(img.size[1]):
```

```
matchIndex = []
pixvalue = imgPix[i, j]
r = eqmatvec[pixvalue[0]]
g = eqmatvec[pixvalue[1] + 256]
b = eqmatvec[pixvalue[2] + 512]

if r in masterRChannel:
    matchIndex.append(masterRChannel.index(r))
else:
    found = False
    n = 1
    while not found and n < 255:
        if (r - n) in masterRChannel:
            found = True
            r = r - n
        elif (r + n) in masterRChannel:
            found = True
            r = r + n
        else:
            n += 1
    matchIndex.append(masterRChannel.index(r))

if g in masterGChannel:
    matchIndex.append(masterGChannel.index(g))
else:
    found = False
    n = 1
    while not found and n < 255:
        if (g - n) in masterGChannel:
            found = True
            g = g - n
        elif (g + n) in masterGChannel:
            found = True
            g = g + n
        else:
            n += 1
    matchIndex.append(masterGChannel.index(g))

if b in masterBChannel:
    matchIndex.append(masterBChannel.index(b))
else:
    found = False
    n = 1
    while not found and n < 255:
        if (b - n) in masterBChannel:
            found = True
            b = b - n
        elif (b + n) in masterBChannel:
            found = True
            b = b + n
        else:
            n += 1
    matchIndex.append(masterBChannel.index(b))
```

```
            resultPix[i,j] = tuple(matchIndex)

    return result

if __name__ == '__main__':
    if len(sys.argv) == 4:
        print "Performing histogram matching of %s against %s...." % \
               (sys.argv[1], sys.argv[2]),
        sys.stdout.flush()
        histogramMatched = HistogramMatchedImage(sys.argv[1], sys.argv[2])
        histogramMatched.save(sys.argv[3])
        print " done!"
        print "Result has been saved in %s" % (sys.argv[3],)

    else:
        print "  Usage: %s <image_to_match> <reference_image> <output_file>"\
                      % (sys.argv[0],)
```

A.3. Trazador de rayos de iluminación directa

```
#include "generalHeader.hpp"
#include "RayTracer.hpp"

RayTracer::RayTracer()
{
objects.clear();
    lightSources.clear();
}

bool RayTracer::OpenOutputDataFile(char *outputfile)
{
  // Open destination file
  if ((df = fopen(outputfile, "wb")) == NULL) {
    printf("Error creating file %s\n", outputfile);
    return false;
  }
  fprintf(df, "##########################################\n");
  fprintf(df, "# YAReW Light Estimation Map Data File     #\n");
  fprintf(df, "#                                          #\n");
  fprintf(df, "# %-49s#\n", outputfile);
  fprintf(df, "##########################################\n");
  fprintf(df, "# Col point  | White level  | Dist to lamp #\n");

  return true;
}

void RayTracer::CloseOutputDataFile()
{
  fclose(df);
}
```

```
Vector RayTracer::Trace(Ray *ray)
{
Vector pos;
//  Triangle* currentObject;

//  for (int i=0; i<objects.length(); i++) {
//  currentObject = (Triangle *) objects[i];
//         currentObject->TestIntersection(ray);
//  }

    bvh.TestIntersection(ray);

if (ray->GetNearestObject() != NULL)
      pos = ray->GetOrigin() + ray->GetDirection() *
                               ray->GetNearestHitDistance();
else
      pos.Set(MAX_DISTANCE,MAX_DISTANCE,MAX_DISTANCE);

return pos;
}


bool RayTracer::LoadSceneFile(char *filename)
{
char token[MAX_LINE_SIZE];
bool inCommentBlock = false;

using namespace std;

ifstream inFile(filename);
inFile >> token;

while (!inFile.eof()) {
if (token[0] != '#') {
if (token[0] == 't') { // Triangle
double P0x, P0y, P0z, P1x, P1y, P1z, P2x, P2y, P2z;
                double noX, noY, noZ;
inFile >> P0x >> P0y >> P0z >> P1x >> P1y >> P1z >> P2x >>
                      P2y >> P2z >> noX >> noY >> noZ;
Triangle *t = new Triangle(Vector(P0x, P0y, P0z), Vector(P1x, P1y, P1z),
   Vector(P2x, P2y, P2z), Vector(noX, noY, noZ));
objects.append(t);
}
else if (token[0] == 'c') { // Camera
              double px, py, pz, lx, ly, lz;
              char name[64];
              inFile >> px >> py >> pz >> lx >> ly >> lz >> name;
              eyePosition.Set (px, py, pz);
              lookTarget.Set(lx,ly,lz);
}
            else if (token[0] == 'l') {
              std::string name, type;
              double locX, locY, locZ, vDirX, vDirY, vDirZ, lightEnergy;
```

```
              inFile >> name >> locX >> locY >> locZ >> type >> vDirX >>
                     vDirY >> vDirZ >> lightEnergy;
              LightSource *l = new LightSource();
              l -> SetName(name);
              l -> SetType(type);
              l -> SetLocation(Vector(locX, locY, locZ));
              l -> SetVDirection(Vector(vDirX, vDirY, vDirZ));
              l -> SetEnergy(lightEnergy);

              lightSources.append(l);
            }
}
inFile.getline(token, MAX_LINE_SIZE, '\n');  // Eat the line
inFile >> token;
}
inFile.close();
    printf("\n\t%d triangles were successfully loaded.\n", objects.length());
    printf("\t%d lamps were successfully loaded.\n", lightSources.length());
    printf("\nGenerating Bounding-Volume Hierarchy data struct....\n");
    bvh = Bvh(objects);

return true;
}

bool RayTracer::RayTrace(bool indoorLightSources)
{
  srand(time(NULL));

  /* Only cast rays from the selected light sources,
     that's indoor or outdoor */
  string lightSourceType;
  if (indoorLightSources)
    lightSourceType = "int"; // Indoor lamps' names start with "int"
  else
    lightSourceType = "ext"; // Outdoor lamps' names start with "ext"

  for (int i = 0; i < lightSources.length(); i++) {
    LightSource *lamp = (LightSource*) lightSources[i];
    if(!lamp->GetName().compare(0, 3, lightSourceType)) {
      int nonNullIntersecPoints = 0;
      int nonVisible = 0;
      int didntHit = 0;

      fprintf(stdout, "Casting rays from %s....\n", lamp->GetName().c_str());
      for (int r = 0; r < NRAYS_TO_TRACE; r++) {
        Ray ray = CastRay(lamp);
        Vector pInt = Trace(&ray);
        if(ray.GetNearestObject() != NULL) {
          Triangle *hitSurface = (Triangle *)ray.GetNearestObject();
          // Check if the hit point is visible from the camera location
          // if(lookTarget.DotProduct(hitSurface->GetVNormal()) < 0) {
            nonNullIntersecPoints += 1;
            /* White level is directly proportional to the energy
               of the light source */
```

```
            double whiteLevel = lamp->GetEnergy() /
                                GetMaxLightEnergy(lightSourceType);
            double pDistToLamp = PointToPointDistance(eyePosition, pInt);
            fprintf(df, "%f\t%f\t%f\t%f\t%f\n", pInt.x, pInt.y, pInt.z,
                                                whiteLevel, pDistToLamp);

            //          }
            //          else
            //            nonVisible += 1;
          }
          else
            didntHit += 1;
        }
        printf("\t%d/%d rays hit some visible surface.\n",
               nonNullIntersecPoints, NRAYS_TO_TRACE);
        printf("\t%d/%d hits were not visible from camera location.\n",
               nonVisible, NRAYS_TO_TRACE);
        printf("\t%d/%d rays didn't hit any surface.\n",
               didntHit, NRAYS_TO_TRACE);
        fprintf(stdout, "Done.\n");
      }
    }
    return true;
}

Ray RayTracer::CastRay(LightSource *lightSource)
{
    Vector lampVNormal = lightSource->GetVDirection();
    Ray rayToCast;
    bool cast = false;

    while(!cast) {
      double x = ((double)rand() / (double)RAND_MAX);
      double y = ((double)rand() / (double)RAND_MAX);
      double z = ((double)rand() / (double)RAND_MAX);
      // Throw dice to make x, y, z coordinates negative or leave them
      // as they are.
      int die = rand() % 2;
      if (die == 1) x *= -1;
      die = rand() % 2;
      if (die == 1) y *= -1;
      die = rand() % 2;
      if (die == 1) z *= -1;

      Vector dir(x, y, z);

      if(dir.Length() <= 1.0) { // Check whether is inside the unit sphere
        // Cast rays in all directions
        if(!lightSource->GetType().compare("Lamp")) {
          rayToCast.SetOrigin(lightSource->GetLocation());
          rayToCast.SetDirection(dir);
          cast = true;
        }
        // Lamp types such as area lights, spots, suns, etc...
```

```
      else {
        /* SPOT SAMPLING */
        // Remove not needed samples
        if(lampVNormal.DotProduct(dir) >= 0.0f) {
          rayToCast.SetOrigin(lightSource->GetLocation());
          rayToCast.SetDirection(dir);
          cast = true;
        }
      }
    }
  }
  return rayToCast;
}

double RayTracer::GetMaxLightEnergy(string lightSourceType)
{
  double maxEnergy = -1.0f;
  for (int i = 0; i < lightSources.length(); i++) {
    LightSource *lamp = (LightSource*) lightSources[i];
    if(!lamp->GetName().compare(0, 3, lightSourceType)) {
      double energy = lamp->GetEnergy();
      if(energy > maxEnergy)
        maxEnergy = energy;
    }
  }
  return maxEnergy;
}

double RayTracer::PointToPointDistance(Vector p1, Vector p2)
{
  Vector diff = p2 - p1;
  return diff.Length();
}
```

A.4. Importador de impactos de iluminación directa

```
#!BPY

import Blender
from Blender import NMesh, Material, Scene, World, Object, Mathutils
from Blender.Mathutils import *
from Blender.Scene import Render

import string, sys, os, time
import dilOptimizer

class IntPoint(object):
    def __init__(self, x, y, z, whiteLevel, distToCamera):
        self.x = x
        self.y = y
        self.z = z
```

```
        self.whiteLevel = whiteLevel
        self.distToCamera = distToCamera

def mulmatvec4x3(a, b):
    # a is vector, b is matrix
r = [0, 0, 0]
r[0] = a[0]*b[0][0] + a[1]*b[1][0] + a[2]*b[2][0] + b[3][0]
r[1] = a[0]*b[0][1] + a[1]*b[1][1] + a[2]*b[2][1] + b[3][1]
r[2] = a[0]*b[0][2] + a[1]*b[1][2] + a[2]*b[2][2] + b[3][2]
return r

def GetPointListFromFile(scene_filename):
    pointList = []
    fd = file(scene_filename, 'r')
    for line in fd.readlines():
        if line[0] != '#':
            words = string.split(line)
            if len(words) == 5:
                x = float(words[0])
                y = float(words[1])
                z = float(words[2])
                wLevel = float(words[3])
                dist = float(words[4])
                p = IntPoint(x, y, z, wLevel, dist)
                pointList.append(p)

    fd.close()
    return pointList

def DarkSceneObjects():
    scene = Blender.Scene.GetCurrent()

    blackMat = Material.New()
    blackMat.rgbCol = [0.0, 0.0, 0.0]
    blackMat.setSpec(0.0)
    blackMat.setRef(0.0)
    for obj in scene.objects:
        if obj.type == 'Mesh':
            data = obj.getData(mesh=True)
            data.materials = [blackMat]

    # Dark world object
    world = World.GetCurrent()
    world.setAmb([0.0, 0.0, 0.0])
    world.setHor([0.0, 0.0, 0.0])
    world.setZen([0.0, 0.0, 0.0])

def ConstructPlane(origin, nvect, diag_length):
    if not(nvect.x == 0 and nvect.y == 0 and nvect.z == 0):
        mesh = NMesh.New()
        a = nvect.x
        b = nvect.y
        c = nvect.z
        d = -DotVecs(origin, nvect)
```

```
        p = None
        if nvect.x != 0:
            px = ((-d) - b - c)/a
            p = Vector(px, 1.0, 1.0)
        elif nvect.y != 0:
            py = ((-d) - a - c)/b
            p = Vector(1.0, py, 1.0)
        else:
            pz = ((-d) - a - b)/c
            p = Vector(1.0, 1.0, pz)

        origin_p = p - origin
        origin_p.normalize()

        u1 = origin_p * (diag_length / 2.0)
        u2 = CrossVecs(origin_p, nvect).normalize() * (diag_length / 2.0)

        p1 = origin + u1
        p2 = origin + u2
        p3 = origin + u1.negate()
        p4 = origin + u2.negate()

        v1 = NMesh.Vert(p1.x, p1.y, p1.z)
        v2 = NMesh.Vert(p2.x, p2.y, p2.z)
        v3 = NMesh.Vert(p3.x, p3.y, p3.z)
        v4 = NMesh.Vert(p4.x, p4.y, p4.z)

        mesh.verts.extend([v1, v2, v3, v4])
        mesh.faces.append(NMesh.Face([v4, v3, v2, v1]))

        return mesh

    return None

def GetCameraObject(scene):
    for obj in scene.objects:
        if obj.type == 'Camera':
            return obj
    return None

def GetCameraLookVect(obj):
    matrix = obj.getMatrix('worldspace')
    obloc = Vector(obj.loc)
    plook = Vector(mulmatvec4x3([0, 0, -1], matrix))
    vlook = plook - obloc
    vlook.normalize()

    return vlook

def GetCameraLocation(obj):
    return [obj.LocX, obj.LocY, obj.LocZ]

def LinkSphereObjectsToScene(pointList):
```

```
    scene = Blender.Scene.GetCurrent()
    obCamera = GetCameraObject(scene)
    camloc = Vector(GetCameraLocation(obCamera))
    camlook = GetCameraLookVect(obCamera)
    camlook.negate()

    diag_length = 0.2

    mesh = NMesh.New()
    mat = Material.New()
    mat.rgbCol = (255, 255, 255)
    mat.mode |= Material.Modes.SHADELESS
    mesh.materials = [mat]
    for p in pointList:
        p = Vector(p.x, p.y, p.z)
        nvect = camloc - p
        m = ConstructPlane(p, nvect, diag_length)
        mesh.verts.extend(m.verts)
        mesh.faces.extend(m.faces)

    obj = Object.New('Mesh')
    obj.link(mesh)
    scene.objects.link(obj)

SCENE_FILENAME = os.getenv('hits_file', -1)
if SCENE_FILENAME == -1:
    print """Error: 'hits_file' environment variable not found.
             You may set it up and retry.\n"""
    sys.exit(1)

print "Adding up photon hits to the blender scene....    "
tinit = time.time()
pointList = GetPointListFromFile(SCENE_FILENAME)
DarkSceneObjects()
LinkSphereObjectsToScene(pointList)
Blender.Redraw()
tend = time.time()
print "Done (%d secs)." % (tend - tinit,)

## Render the scene
scene = Scene.GetCurrent()
renderContext = scene.getRenderingContext()

print "Rendering blender scene...."
tinit = time.time()
renderContext.setRenderer(Render.INTERNAL)
renderContext.setRenderPath(os.getcwd() + "/")
renderContext.setImageType(Render.PNG)
renderContext.enableOversampling(False)
renderContext.enableRayTracing(False)
renderContext.enableShadow(False)
renderContext.enableEnvironmentMap(False)
renderContext.sizePreset(Render.PC)
renderContext.render()
```

```
tend = time.time()
print "Done (%d secs)." % (tend - tinit,)
```

A.5. Optimizador genético de configuraciones render

```
#!/usr/env/bin python

import orange, random


class HeapList(object):
    def __init__(self, length):
        self.length = length
        self.elements = {}

    def __getElement(self, value):
        keys = self.elements.keys()
        for k in keys:
            if self.elements[k] == value:
                return k
        return None

    def addElement(self, element, value):
        self.elements[element] = value
        if len(self) > self.length:
            values = self.elements.values()
            values.sort()
            minValue = values[0]
            worstElement = self.__getElement(minValue)
            del self.elements[worstElement]

    def popRandomElement(self):
        keys = self.elements.keys()
        r = random.randint(0, len(keys) - 1)
        el = keys[r]
        val = self.elements[el]
        del self.elements[el]
        return el, val

    def getRandomElement(self):
        keys = self.elements.keys()
        r = random.randint(0, len(keys) - 1)
        el = keys[r]
        val = self.elements[el]
        return el, val

    def __str__(self):
        s = ''
        for k in self.elements.keys():
            s += str(k) + ' (' + str(self.elements[k]) + ')\n'
        return s
```

```
    def __len__(self):
        return len(self.elements.keys())


def _fitnessFunction(renderTime, renderQuality):
    renderTime = float(renderTime.value)
    renderQuality = float(renderQuality.value)
    renderQuality *= 10.0

    return (renderQuality / renderTime)

def _createBestDatasetExamplesList(dsTime, dsQuality, length=10):
    exList = HeapList(length)

    for i in range(len(dsTime)):
        exTime = dsTime[i].getclass()
        exQuality = dsQuality[i].getclass()
        value = _fitnessFunction(exTime, exQuality)
        exList.addElement(i, value)

    return exList

def _createUserExampleNearestNeighborsList(exTime, exQuality,
                                           dsTime, dsQuality, length=10):
    nnc = orange.FindNearestConstructor_BruteForce()
    nnc.distanceConstructor = orange.ExamplesDistanceConstructor_Euclidean()
    did = orange.newmetaid()
    nn = nnc(dsQuality, 0, did)

    k = length * 2
    l = HeapList(length)
    for ex in nn(exQuality, k):
        for i in range(len(dsQuality)):
            if ex == dsQuality[i]:
                exTime = dsTime[i].getclass()
                exQuality = dsQuality[i].getclass()
                value = _fitnessFunction(exTime, exQuality)
                l.addElement(i, value)
                break
    return l

def _selectRandomParents(exTime, exQuality, dsTime,
                         dsQuality, userAffinity=0.5, listLength=10):
    # List with the best examples available in the dataset
    bestDsExamples = _createBestDatasetExamplesList(dsTime,
                                                    dsQuality, listLength)
    # List with the best examples in dataset which are most
    # similar to the user's
    userSimilarExamples = _createUserExampleNearestNeighborsList(exTime,
                                  exQuality, dsTime, dsQuality, listLength)

    # Take elements from each list depending on the specified userAffinity
    userElements = int(listLength * userAffinity)
```

```
    dsElements = listLength - userElements

    # Create parents heap list
    parents = HeapList(listLength)
    # Take examples from user's nearest neighbors examples
    for i in range(userElements):
        el, val = userSimilarExamples.popRandomElement()
        parents.addElement(el, val)
    # Take examples from the best dataset examples list
    for j in range(dsElements):
        el, val = bestDsExamples.popRandomElement()
        parents.addElement(el, val)

    # Sometimes, when both lists (best nearest neighbors and
    # best dataset examples) share some examples, the parents'
    # list remain with less than listLength elements.
    while len(parents) != listLength:
        el, val = userSimilarExamples.popRandomElement()
        parents.addElement(el, val)
        el, val = bestDsExamples.popRandomElement()
        parents.addElement(el, val)

    return parents

def _reproductiveAssistance(exFather, exMother):
    exSon = orange.Example(exFather.domain)
    # Avoid class attribute
    splitPoint = random.randint(0, len(exSon.domain) - 2)
    for i in range(splitPoint):
        exSon[i] = exFather[i]
    # Leave class value as unknown '?'
    for i in range(splitPoint, len(exSon.domain) - 1):
        exSon[i] = exMother[i]

    return exSon

def _mutantValueForContinuousAttr(attr, dsQuality):
    # Backup dataset
    data = orange.ExampleTable(dsQuality.domain, dsQuality[:])
    data.sort([attr]) # Sorting of backup dataset examples based on attr

    minAttrValue = data[0][attr]
    maxAttrValue = data[len(data) - 1][attr]
    attrValue = random.uniform(minAttrValue, maxAttrValue)
    if (data[0][attr].value - int(data[0][attr].value)) == 0.0:
        return float(int(attrValue))
    return attrValue

def _mutate(example, dsQuality, prob=0.05):
    r = random.random()
    if r < prob:
        domain = example.domain
        # Avoid mutation of class value
        attrIndex = random.randint(0, len(example.domain) - 2)
```

```
        if domain[attrIndex].varType is orange.VarTypes.Discrete:
            example[attrIndex] = (int(example[attrIndex]) + \
                                  random.randint(0, len(domain))) % \
                                  len(domain[attrIndex].values)
        elif domain[attrIndex].varType is orange.VarTypes.Continuous:
            example[attrIndex] = _mutantValueForContinuousAttr \
                                  (domain[attrIndex], dsQuality)
        else: # Other type of attribute (not discrete and not continuous)
            pass
    return example

def _runIter(exTime, exQuality, dsTime, dsQuality, parents):
    # Select examples to combine for generating the new one
    exIndexFather, exValueFather = parents.getRandomElement()
    exFatherQuality = dsQuality[exIndexFather]
    exFatherTime = dsTime[exIndexFather]
    # Do not allow reproduction of the same father and mother
    exIndexMother = exIndexFather
    while exIndexMother == exIndexFather:
        exIndexMother, exValueMother = parents.getRandomElement()
        exMotherQuality = dsQuality[exIndexMother]
        exMotherTime = dsTime[exIndexMother]

    # Generate new example from the combination of father
    # and mother examples
    exSon = _reproductiveAssistance(exFatherQuality, exMotherQuality)
    mutantSon = _mutate(exSon, dsQuality, prob=0.1)

    # Generated example has quality dataset based domain.
    # Let's create also the time dataset based example.
    mutantSonQuality = mutantSon
    #Create empty time-based domain example
    mutantSonTime = orange.Example(exFatherTime.domain)
    # Fill all attribute values but the class one
    for attr in mutantSonTime.domain.attributes:
        mutantSonTime[attr.name] = mutantSonQuality[attr.name]

    return (mutantSonTime, mutantSonQuality)

def _evalIter(exTime, exQuality, timeEstimator, qualityClassifier):
    timeEst = timeEstimator.Estimate(exTime, method='knn')
    qualityClass = qualityClassifier.Classify(exQuality, method='knn')
    fitnessValue = _fitnessFunction(timeEst, qualityClass)

    return fitnessValue

def GeneticOptimizer(userScene, timeEstimator, qualityClassifier, nIter=100,
                     userAffinity=0.5, timeEstMethod='knn',
                     qualityClassMethod='knn', parentsLength=10):
    random.seed(None)

    time_ex = userScene.GetTimeDomainExample()
    quality_ex = userScene.GetQualityDomainExample()
    time_ds = timeEstimator.GetDataset()
```

```
quality_ds = qualityClassifier.GetDataset()

# Create list of parents basing on best dataset examples
# and user config nearest neighbors
parents=_selectRandomParents(time_ex,quality_ex,time_ds,quality_ds,
                             userAffinity,parentsLength)

optValue = _evalIter(time_ex, quality_ex, timeEstimator,
                     qualityClassifier)
optTimeEx = time_ex
optQualityEx = quality_ex

for i in range(nIter):
    timeSon, qualitySon = _runIter(time_ex, quality_ex, time_ds,
                                   quality_ds, parents)
    sonValue = _evalIter(timeSon, qualitySon, timeEstimator,
                         qualityClassifier)
    if sonValue > optValue:
        optValue = sonValue
        optTimeEx = timeSon
        optQualityEx = qualitySon
        # Add new example to datasets and list of potential parents
        qualitySon.setclass(qualityClassifier.Classify(qualitySon))
        quality_ds.append(qualitySon)
        timeSon.setclass(timeEstimator.Estimate(timeSon))
        time_ds.append(timeSon)
        el, val = len(quality_ds) - 1, sonValue
        parents.addElement(el, val)

return (optTimeEx, optQualityEx)
```

Apéndice B

Escenas Incluidas en la Base de Conocimiento

Figura B.1: Render de las escenas incluidas en la base de conocimiento

Figura B.2: Render de las escenas incluidas en la base de conocimiento (*Continuación*)

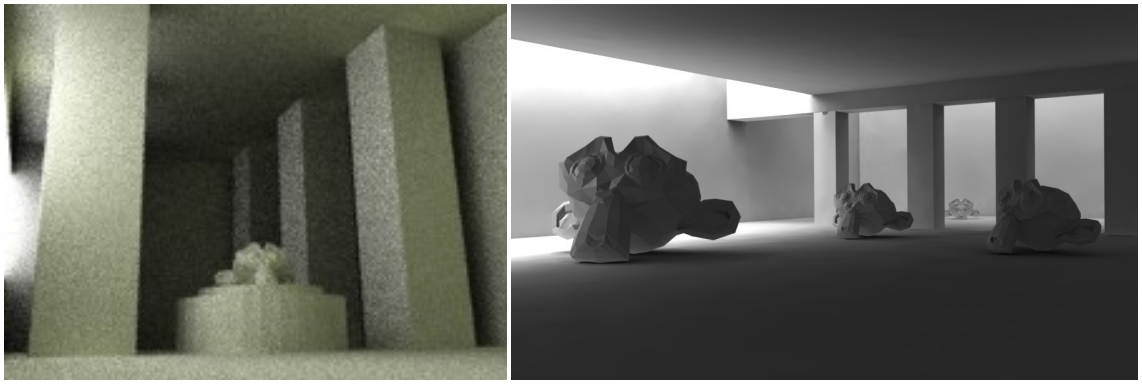
Figura B.3: Render de las escenas incluidas en la base de conocimiento (*Continuación*)

Figura B.4: Render de las escenas incluidas en la base de conocimiento (*Continuación*)

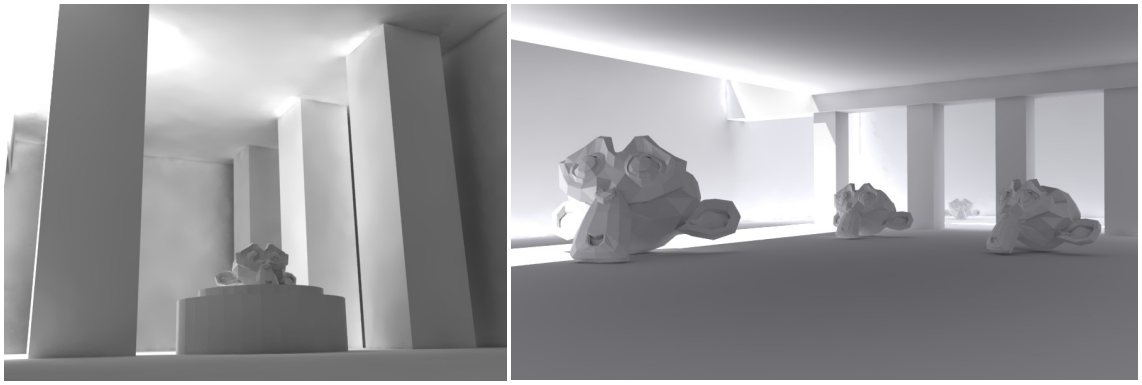
Apéndice C

Imágenes a Color

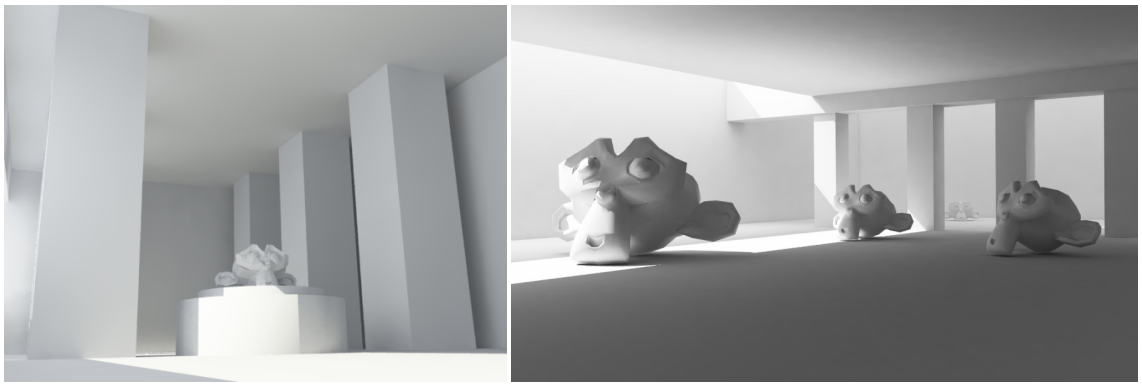
Figura C.1: Resultados de la misma escena con diferentes métodos de render: a) RayCasting b) RayTracing (Whitted) c) Ambient Occlusion d) PathTracing (Skydome) e) PathTracing (Un foco de iluminación directa) f) PathTracing + Photon Mapping (+ Iluminación basada en imagen HDR).



(a) Resultados de baja calidad



(b) Resultados de media calidad



(c) Resultados de alta calidad

Izquierda: Escena “Simple”. **Derecha:** Escena “Compleja”.

Figura C.2: Algunos resultados de diferente calidad obtenidos en el experimento sobre configuración de render realizado.

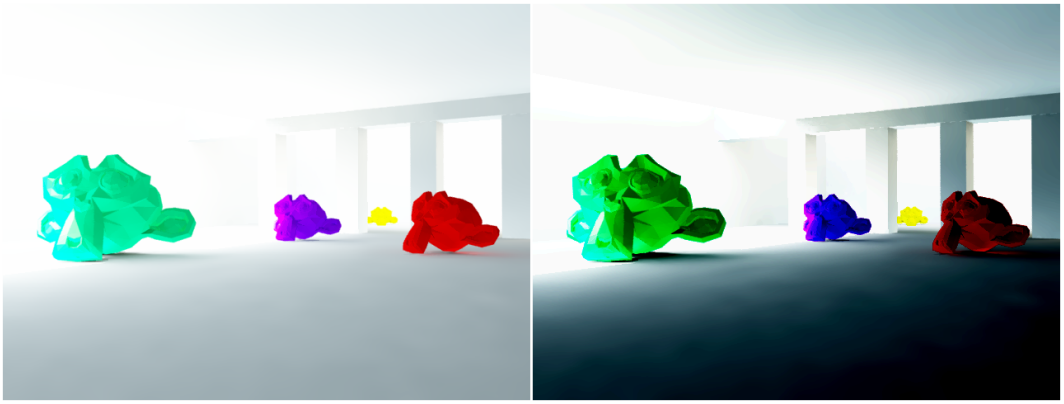
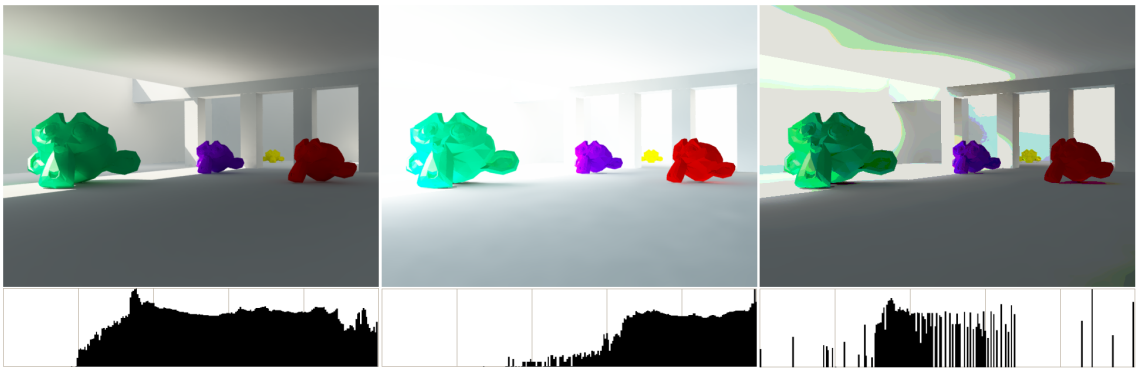


Figura C.3: Efecto de aplicar el método de ecualización de histograma sobre una imagen: **Izquierda**: Imagen original, **Derecha**: Imagen ecualizada.



(a) Imagen de referencia (b) Imagen a comparar (c) Imagen resultado

Figura C.4: Ejemplo de imagen procesada mediante proyección de histograma.

Raydepth	OSA	GIQuality	Depth	CDepth	PhCount	PhRadius	PhMixCount	ShQuality	Prec	Refine	EPower	GIPower
5	16	HIGH	3	1	100000	1.0	100	0.9	10	1.0	1.0	1.0

(a) Configuración de usuario establecida para las pruebas

(b) Imagen producida a raíz de la realización del render con la configuración expuesta arriba

Figura C.5: **Arriba**: Configuración de parámetros render establecida para la fase de pruebas. **Abajo**: Resultado de realizar el render de la escena de pruebas con la configuración de parámetros indicada en (a).

(a) Resultado con un nivel de optimización *bajo*

(b) Resultado con un nivel de optimización *medio*

(c) Resultado con un nivel de optimización *alto*

(d) Resultado con un nivel de optimización *muy alto*

Nivel de Optimización	Raydepth	OSA	GIQuality	Depth	CDepth	PhCount	PhRadius	PhMixCount	ShQuality	Prec	Refine	EPower	GIPower
Bajo	3	16	HIGH	1	5	27485	10.9	67	0.173	38	0.662	12.5	0.14
Medio	32	11	BEST	1	5	100000	1.0	100	0.173	38	0.662	12.5	0.14
Alto	23	5	HIGH	1	5	27485	2.62	146	0.173	38	0.662	20.5	0.14
Muy Alto	26	11	BEST	1	5	27485	13.02	54	0.151	39	0.467	1.9	5.28

(e) Configuraciones de parámetros render sugeridas según el nivel de optimización empleado

Figura C.6: Imágenes producidas al realizar el render de la escena de prueba con las distintas configuraciones sugeridas por el sistema con un valor de afinidad 0.5

(a) Resultado con un nivel de optimización *bajo*

(b) Resultado con un nivel de optimización *medio*

(c) Resultado con un nivel de optimización *alto*

(d) Resultado con un nivel de optimización *muy alto*

Nivel de Optimización	Raydepth	OSA	GIQuality	Depth	CDepth	PhCount	PhRadius	PhMixCount	ShQuality	Prec	Refine	EPower	GIPower
Bajo	36	8	BEST	1	5	273863	33.7	454	0.312	39	0.564	0.23	13.54
Medio	36	5	LOW	1	6	100927	1.9	193	0.185	44	0.752	0.58	11.13
Alto	36	8	BEST	6	1	96634	1.9	384	0.185	38	0.752	0.58	11.13
Muy Alto	28	11	BEST	1	5	96634	1.9	193	0.185	44	0.214	0.58	11.13

(e) Configuraciones de parámetros render sugeridas según el nivel de optimización empleado

Figura C.7: Imágenes producidas al realizar el render de la escena de prueba con las distintas configuraciones sugeridas por el sistema con un valor de afinidad 0.0

Bibliografía

[1] Open Source Initiative's Open Source Definition. http://www.opensource.org/docs/osd.

[2] Orange Data Mining Framework Official Website. http://www.ailab.si/orange/.

[3] PHP-Hypertext Processor Official Website. http://www.php.net/.

[4] Sitio Web Oficial de Blender. http://www.blender.org/.

[5] Sitio Web Oficial de Magarro. http://www.boxel.info/magarro/.

[6] Sitio Web Oficial de Yafray. http://www.yafray.org/.

[7] Sitio Web Oficial del lenguaje de programación Python. http://www.python.org/.

[8] Alpaydin, E. *Introduction to Machine Learning*. The MIT Press, 2004. ISBN: 0-262-01211-1.

[9] Appel, A. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of Spring Joint Computer Conference*, number 32, pages 37–49, April 1968.

[10] Bentley, J.L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975. ISSN: 0001-0782.

[11] Bouknight, W. A procedure for generation of three-dimensional half-toned computer graphics presentations. *Communications of the ACM*, 13(9):527–536, September 1970. ISSN: 0001-0782.

[12] Bratko, I. *Prolog Programming for Artificial Intelligence*. Addison Wesley, 3rd edition, 2000. ISBN-13: 978-0201403756.

[13] Cook, R., Porter, T., Carpenter, L. Distributed Ray Tracing. In *Proceedings of the 11th annual conference on Computer Graphics and Interactive Techniques*, pages 137–145, 1984. ISSN: 0097-8930.

[14] Ferwerda, J. Three varieties of realism in computer graphics. In *Proceedings SPIE Human Vision and Electronic Imaging*, pages 290–297, 2003.

[15] Glassner, A. *An Introduction to Ray Tracing*. Morgan Kaufmann Series in Computer Graphics, 1989. ISBN-13: 978-0122861604.

[16] Goldsmith, J., Salmon, J. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987. ISSN: 0272-1716.

[17] Gonzalez, R., Woods, R. *Digital Image Processing*. Prentice Hall, 2nd edition, 2002. ISBN-13: 978-0201180756.

[18] Goral, C., Torrance, K., Greenberg, D., Battaile, B. Modeling the Interaction of Light between Diffuse Surfaces. In *Proceedings of the 11th annual conference on Computer Graphics and Interactive Techniques*, pages 213–222, 1984. ISSN: 0097-8930.

[19] Jensen, H.W. *Realistic Image Synthesis Using Photon Mapping*. AK Peters, 2001. ISBN: 1568811470.

[20] Kajiya, J. The rendering equation. In *Proceedings of the 13th annual conference on Computer Graphics and Interactive Techniques*, pages 143–150, 1986. ISSN: 0097-8930.

[21] Lafortune, E., Willems, Y. Bi-directional Path Tracing. In *Proceedings of the 3rd International Conference on Computational Graphics and Visualization Techniques*, pages 145–153, 1993.

[22] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1996. ISBN: 978-3-540-60676-5.

[23] Mitchell, T.M. *Machine Learning*. McGraw-Hill, international edition, 1997. ISBN: 0-07-115467-1.

[24] Myszkowski, K., Tawara, T., Akamine, H., Seidel, H. Perception-guided global illumination solution for animation rendering. In *Proceedings of the 28th annual conference on Computer Graphics and Interactive Techniques*, pages 221–230, 2001. ISBN: 1-58113-374-X.

[25] Quinlan, J.R. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, Marzo 1986. ISSN: 0885-6125 (Impreso) 1573-0565 (Online).

[26] Quinlan, J.R. *C4.5: Programs for machine learning*. Morgan Kaufmann series in machine learning, 1993. ISBN: 1-55860-238-0.

[27] Russell, S., Norvig, P. *Inteligencia Artificial. Un enfoque Moderno*. Pearson Educación, 2nd edition, 2004. ISBN: 84-205-4003-X.

[28] Sierra Araujo, B. *Aprendizaje Automático: conceptos básicos y avanzados*. Pearson Educación, 2006. ISBN: 84-8322-318-X.

[29] Tan, P., Steinbach, M., Kumar, V. *Introduction to Data Mining*. Pearson - Addison Wesley, 2006. ISBN: 0-321-42052-7.

[30] Veach, E., Guibas, L. Metropolis Light Transport. In *Proceedings of the 24th annual conference on Computer Graphics and Interactive Techniques*, pages 65–76, 1997. ISBN: 0-89791-896-7.

[31] Whitted, T. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980. ISSN: 0001-0782.

[32] Witten, I.H., Frank, E. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann series in data management systems, 2nd edition, 2005. ISBN: 0-12-088407-0.

[33] Zhukov, S., Iones, A., and Kronin, G. An Ambient Light Illumination Model. In *Proceedings of the Eurographics Workshop on Rendering*, pages 45–55, 1998.