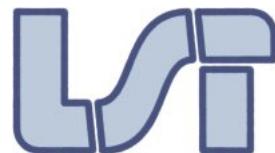


STIPPLING



Universidad de Granada



Autor:

José Ignacio Carmona Villegas
joseicv@correo.ugr.es

Tutor:

Germán Arroyo Moreno
arroyo@ugr.es

Curso 2013/14, convocatoria de Junio

«El sabio sabe que ignora.»

Confucio

Motivación

El conocimiento me fascina, y desde siempre, aprender ha sido mi mayor afición. Por eso, cuando Juan Carlos Torres Cantero me propuso hacer unas prácticas en el Laboratorio de Realidad Virtual, no dudé en aceptar la oportunidad. El hecho de poder participar en la expansión del conocimiento humano, aunque sea remóticamente, me atrae.

Este proyecto continúa una de las líneas de trabajo del equipo de investigación del Laboratorio de Realidad Virtual de la Universidad de Granada en el área de la Renderización No Fotorrealista (NPR, Non-Photorealistic Rendering), y consiste en el desarrollo de una prueba de concepto de una idea propuesta por Germán Arroyo Moreno.

Con este proyecto pretendo colaborar en el intento de expandir la frontera del conocimiento. Aún cuando el intento sea tan humilde que apenas logremos modificar infinitesimalmente la tangente de la superficie del hipotético volumen que representa el conocimiento humano.

Agradecimientos

Quiero agradecer a mi familia el apoyo que siempre me han prestado, especialmente cuando me hacían ver que mi inversión excesiva de tiempo en el proyecto ha supuesto una merma para mi salud. Sé que tengo que dormir y desvelarme trabajando no es excusa. Quiero agradecer a mis amigos su constante apoyo, pues a menudo creen en mí más que yo mismo.

Me gustaría también agradecer el apoyo de los miembros de los equipos del Laboratorio de Realidad Virtual de la Universidad de Granada, así como de la empresa Virtum Graphics, por haber estado siempre dispuestos a responder mis preguntas e incluso a intentar implicarme en otros proyectos de mayor calado. He de agradecer especialmente la atención que Luis López Escudero ha tenido siempre para conmigo durante mi período de prácticas en el Laboratorio de Realidad Virtual, y a Germán Arroyo Moreno, por haber sido capaz de soportar mis interminables interrogatorios siendo mi tutor durante el desarrollo del proyecto.

Finalmente, quisiera recordar a todas las personas que he tenido el placer o la desgracia de conocer, pues su paso por mi vida me ha ayudado a convertirme la persona que soy.

A todos, ¡Muchas gracias!

Índice general

1. Introducción	7
2. Objetivos	9
2.1. Creación de una aplicación multiplataforma	9
2.2. Interfaz gráfica de usuario orientada a ser usada por artistas .	10
2.3. Reconstrucción de escena	10
2.4. Motor 3D con capacidad de creación de modelos complejos e iluminación	11
2.5. Mejora de los algoritmos de punteado mediante el uso de información tridimensional	12
3. Trabajos previos	14
4. Descripción de Stippling	18
5. Planificación.	23
5.1. Planificación inicial	23
5.2. Desarrollo del proyecto e incidencias	25
6. Análisis	27
7. Diseño	31
7.1. Diseño estructural	31
7.2. Diseño de interacciones	35
7.2.1. Posicionamiento de un modelo	35
7.2.2. Rotación de un modelo en torno al eje Y	36
7.2.3. Redimensionado de altura	36
7.2.4. Redimensionado del radio	38
7.2.5. Redimensionado de caras planas	38

8. Algoritmos usados	40
8.1. Reconstrucción de escena a partir de metadatos de una fotografía	40
8.2. Organización de puntos mediante un QuadTree	43
9. Recursos, resultados y pruebas	45
9.1. Recursos	45
9.2. Resultados y pruebas	46
10. Manual de usuario	68
11. Manual de compilación	75
12. Problemas y soluciones	78
12.1. Degeneración de poliedros de CGAL	78
12.2. Integración de VBO/VAO y el cauce antiguo de OpenGL . . .	78
12.3. Mejora de la eficiencia de representación de la imagen de punteado	79
12.4. Uso de renderización en textura para evitar realizar renderizaciones redundantes	79
12.5. Uso de múltiples texturas para evitar aberraciones producidas por la interpolación realizada sobre texturas.	79
12.6. Uniformidad del conjunto de normales calculado por CGAL .	80
12.7. Concurrencia de contextos de OpenGL	81
13. Mejoras y líneas de trabajo futuras	82
14. Conclusiones	85
14.1. Conclusiones referentes a los objetivos	85
14.2. Valoración personal	86
Bibliografía	88
Apéndice A. Recorridos de árboles	92
A.1. Primero en anchura	92
A.2. Primero en profundidad	92
Apéndice B. Umbralización	94
Apéndice C. Tramado	95
C.1. Floyd-Steinberg	96
C.2. Stucki	96

Apéndice D. Código ajeno, modificado o incluido	98
D.1. Frame Buffer Objects	98
D.2. Soporte de la biblioteca CGAL	99

Capítulo 1

Introducción

La creación de arte digital está experimentando una expansión y aceptación entre el público sin precedentes, y este entorno propicia la aparición de herramientas digitales de creación total o parcial de arte. Tal como explica Winkenbach [35], hay ocasiones en que el renderizado no fotorealista es más conveniente. A veces, para comunicar información particularmente rica o compleja, una abstracción nos ayuda mucho más que una representación realista y sobrecargada de detalles. Esto es especialmente cierto en diseños técnicos, pero también en el arte, puesto que el artista puede resaltar más unos detalles sobre otros, para que su creación exprese aquello que concibió en su mente.

La Renderización No Fotorealista es un área de estudio de la Informática Gráfica que se centra en el estudio y desarrollo de herramientas y métodos para la creación de arte digital y representaciones abstractas.

Con este objetivo, se abren varias líneas de investigación en el Laboratorio de Realidad Virtual de la Universidad de Granada (LRV-UGR), que pretenden desarrollar la técnica de Punteado (Stippling) digital mediante la creación de algoritmos y herramientas que faciliten el proceso creativo de los artistas, siendo una de ellas a la que este proyecto se adscribe.

La técnica de Punteado consiste en la creación de patrones que simulen distintos grados de solidez mediante el uso de pequeños puntos. Este tipo de patrones tienen un origen natural (pueden ser observados en pétalos y sépalos de plantas en flor), y sus efectos son frecuentemente emulados por artistas.

Esta técnica es ampliamente usada, no sólo con fines artísticos, sino también con fines científico-técnicos, pues, por ejemplo, se usa de forma habitual en botánica para representar las características principales de una especie de

flora, resaltando especialmente las características que la distinguen de otras especies similares.

Este proyecto es una prueba de concepto de una idea propuesta por Germán Arroyo Moreno, enmarcada dentro de las líneas de investigación que el Laboratorio de Realidad Virtual mantiene abiertas en el área de NPR, concretamente en el área de Stippling.

Antes de comenzar, conviene establecer la estructura del documento. El primer paso es, por supuesto, dejar claros los objetivos del proyecto, para continuar repasando los trabajos previos a este proyecto, resaltando su relación con el mismo y su importancia.

Seguidamente, se describirá el algoritmo de Stippling que se implementa, detallando las mejoras con respecto a trabajos anteriores. Como en todo proyecto, se establecerá una planificación, un ligero análisis del problema y se expondrá el diseño establecido, indicando la motivación del mismo.

A continuación se detallarán los algoritmos que se han usado como apoyo, los recursos que se han dispuesto para el desarrollo del proyecto, los resultados y pruebas obtenidos y los manuales de usuario y de compilación.

Finalmente, se explicarán los problemas más significativos que han aparecido durante el desarrollo, y se explicarán las soluciones tomadas, se expondrán las posibles líneas de investigación y desarrollo futuras relacionadas con el proyecto, se extraerán las conclusiones del trabajo realizado y se adjuntarán la bibliografía y los anexos necesarios para mejor comprensión del proyecto.

Capítulo 2

Objetivos

El objetivo del proyecto consiste en crear una aplicación multiplataforma que pretende mejorar los algoritmos anteriormente publicados, capaz de crear una imagen indistinguible de la que un artista crearía mediante el método de punteado. El componente innovador de este proyecto consiste en permitir al usuario trabajar con información 3D y asociarla a la fotografía que estaba visualizando, y de esta forma mejorar la información extraída de la propia imagen 2D, permitiendo así mejorar el proceso de punteado y obtener un resultado más detallado y realista.

Para entender mejor los componentes diseñados en software, se procede a especificar y detallar las partes del proyecto separando este objetivo principal en en objetivos más pequeños.

2.1. Creación de una aplicación multiplataforma

El objetivo de esta sección es:

- Crear una aplicación multiplataforma que opere indistintamente bajo Linux y Windows.

Hoy en día, la necesidad de que el software sea soportado por múltiples plataformas es algo patente. Para lograr este objetivo, Germán Arroyo sugirió el uso de la herramienta CMake, que permite generar ficheros de compilación para múltiples plataformas.

2.2. Interfaz gráfica de usuario orientada a ser usada por artistas

En esta sección, el objetivo pasa por desarrollar una interfaz gráfica de usuario capaz de:

- Disponer de menús sencillos y descriptivos.
- Estar organizada de forma que las herramientas más relevantes sean visibles y sencillas de utilizar.

Dado que el dominio de usuarios de la aplicación es muy específico, la aplicación deberá ser tan sencilla y fácilmente identifiable como sea posible, manteniendo las herramientas disponibles en la pantalla y siguiendo las recomendaciones que Germán Arroyo establece por su conocimiento de primera mano del grupo de usuarios.

2.3. Reconstrucción de escena

Los objetivos de esta sección son:

- Calcular el campo de visión de que disponía la cámara en el momento de la toma de la fotografía.
- Estimar la distancia a la que estaría el objetivo fotografiado.

Para poder utilizar el motor 3D y permitir al usuario construir los modelos tridimensionales que considere más importantes, el primer paso es reconstruir la escena tal como era en el momento de la toma de la fotografía. Para ello, y ya que a partir de una sola fotografía es imposible poder realizar deducciones acerca de la dimensión de profundidad, se deberá recurrir a los metadatos que la cámara incluye en la fotografía. En caso de que dichos metadatos no estén incluidos, se deberá permitir al usuario definir los parámetros que definen la escena. Además el modelo de cámara usada (también disponible como metadato) es también necesario para definir la abertura. Por supuesto, una

vez conocidos los metadatos mencionados, y calculado el campo de visión, se podrá realizar una estimación de la distancia mediante aplicación de trigonometría.

2.4. Motor 3D con capacidad de creación de modelos complejos e iluminación

En esta sección, el objetivo pasa por desarrollar un motor 3D capaz de:

- Crear y editar modelos simples.
- Crear modelos complejos a partir de otros mediante operaciones booleanas (CSG).
- Permitir modificar la posición de los modelos y su orientación.
- Proporcionar múltiples opciones de renderización de los modelos.
- Guardar y cargar el árbol CSG de modelos.

Como objetivo, se necesita desarrollar un motor 3D capaz de permitir al usuario la creación y modificación de sólidos primitivos, no sólo permitiendo modificar su geometría, sino también su posición y orientación. Además se deberá permitir la creación y modificación de sólidos complejos (por composición a partir de sólidos primitivos), de nuevo permitiendo modificar tanto su geometría como su posición y orientación.

El motor deberá ser lo bastante preciso como para permitir crear los modelos necesarios, es decir, que se puedan editar y mantener mientras se mantiene una vista en perspectiva, de forma que puedan corresponderse a la escena representada por la fotografía.

Además, el motor deberá poder renderizar los sólidos de varias formas. La primera de ellas debe facilitar su edición, para lo cual se usará una estructura de alambres. Será necesario también que el motor pueda renderizar los sólidos sin iluminación, con color de superficie en profundidad, de forma que se pueda observar fácilmente la posición de los elementos. El hecho de que los sólidos se rendericen con iluminación ayuda al usuario en la construcción del modelo, y posibilita que se observen fácilmente las aristas del modelo, distinguiéndolas de los cambios de contraste creados por la iluminación en la imagen original.

2.5. Mejora de los algoritmos de punteado mediante el uso de información tridimensional

En esta sección, el objetivo pasa por desarrollar un motor de punteado capaz de:

- Puntar la imagen proporcionada.
- Proporcionar opciones de punteado generales que permitan variar la imagen punteada resultante.
- Mejorar el algoritmo de punteado mediante el aprovechamiento de la información tridimensional que el árbol CSG proporciona.
- Proporcionar opciones de punteado específico, que permitan puntar zonas (correspondientes a modelos del árbol CSG) de forma independiente.
- Generar una imagen final punteada en disco.

Es necesario desarrollar un motor de stippling que, además de las capacidades básicas de punteado, aproveche la información del modelo tridimensional.

Para lograrlo, el motor deberá ser capaz de generar imágenes punteadas a partir de la imagen original y una serie de parámetros definidos por el usuario. El primer paso deberá ser aplanar la imagen a un solo canal (nivel de gris), a continuación, se reducen los niveles de gris a sólo dos, y finalmente, se generan puntos a partir de dicha imagen.

Para lograr que la imagen generada sea más realista, se deberá aplicar un pequeño ruido (controlable por el usuario) a la posición de los puntos, para así ocultar mejor su origen computacional. Las mejoras que aprovechan el motor tridimensional consistirán en el uso de la información proporcionada por los bordes del modelo para evitar que los puntos correspondientes a dichos bordes sufran las modificaciones de posición del ruido antes mencionado, ya que los artistas, tienden a utilizar trazos lineales en los contornos de las figuras (a base de puntos).

Finalmente, como podemos observar en imágenes creadas por artistas, la mayor parte de los puntos se invierten en el objeto principal, y el resto no

se puntean con tanto detalle. Es por esta razón que el motor de stippling deberá dar más peso a los puntos contenidos en el área modelada durante la generación de puntos.

Capítulo 3

Trabajos previos

Este proyecto se enmarca dentro del área de la Informática Gráfica (Computer Graphics, CG), y más concretamente, dentro del área de Non-Photorealistic Rendering (NPR, campo muy trabajado por Bruce y Amy Gooch [8]), también conocida como visualización expresiva. En la Informática Gráfica, existen principalmente dos grandes corrientes clasificadas atendiendo al realismo de las renderizaciones, la corriente tradicional, fotorrealista (Photorealistic Rendering, PR), y la no fotorrealista (Non-Photorealistic Rendering, NPR).

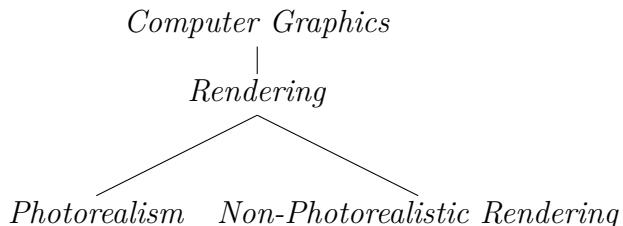


Figura 3.1: Contexto de la Informática Gráfica en que se sitúa el área de NPR.

Enmarcados en el área de NPR, hay múltiples métodos y algoritmos, con diversos objetivos y se mencionan para dar contexto al proyecto. De hecho, el proyecto está enfocado en un grupo y subcampo específico, el grupo de métodos que actualmente aún halftoning y stippling, prestando especial atención al objetivo final, emular la técnica de punteado.

Entre los grupos de técnicas y algoritmos propios del área de NPR (representadas en la figura 3.2), el grupo de Painterly mantiene una relación especial con el grupo de técnicas de stippling, pues posee técnicas destinadas

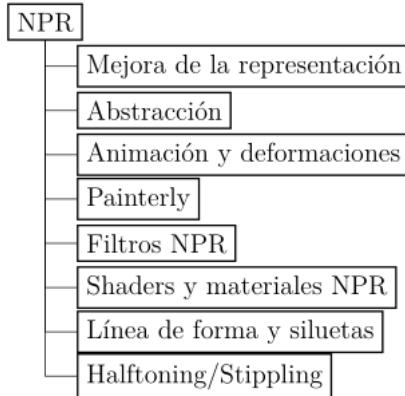


Figura 3.2: Contexto en que se sitúa el grupo de métodos de Halftoning/Stippling.

a simular la corriente artística del Puntillismo, pero debido a las grandes diferencias existentes entre ambas técnicas (en puntillismo, los puntos tienen color, se puntea toda la escena, los puntos no comparten las mismas propiedades que los puntos de stippling ya que los puntos de stippling están orientados a impresión, etc.), el grupo de técnicas de Halftoning es el más apropiado.

El conjunto de técnicas de Halftoning, en que se incluye la técnica de stippling que implementa el proyecto, pretende, partiendo de una imagen original, crear otra versión a base de puntos, de forma que se simulen las formas y texturas representadas en la imagen original.

Las técnicas de stippling, pretenden especificar la técnica de halftoning de forma que los puntos sean similares a los puntos realizados por un artista con una pluma y tinta sobre papel, y que el resultado final emule el trabajo completo del artista, incluyendo no solo la simulación de formas y texturas, sino también de los recursos expresivos de que dispone.

Aunque este proyecto se basa en algunos de los trabajos desarrollados por el Laboratorio de Realidad Virtual ([2], [20] y [21]), son muchos los autores que han trabajado para lograr una mejor emulación de la técnica de punteado. Tradicionalmente, los trabajos se centraban en algoritmos de punteado que partían de una entrada bidimensional (imagen) o de una entrada tridimensional (modelos). Este proyecto trata de combinar ambas perspectivas, por lo que el espectro de trabajos relacionados es muy grande.

Entre los trabajos que requieren una entrada tridimensional, cabe destacar

los trabajos centrados en puntear datos volumétricos ([17], [18]), los métodos de punteado para superficies implícitas ([7], [26]), los métodos de punteado de superficies poligonales ([19], [31]), los métodos de animación de superficies punteadas ([24], [33]) o la computación de los puntos en un dominio híbrido de geometría e imagen ([38]).

Entre los trabajos que requieren una entrada bidimensional, podemos destacar los trabajos basados en renderización basada en pinceladas ([11]), los trabajos basados en computación de una distribución deseable de puntos mediante la computación del diagrama centroidal de Voronoi ([16], [22], [6], [28]), los métodos de punteado basados en sistemas multiagente ([25]), métodos de punteado que utilizan distribuciones no uniformes ([23], [14], [15]).

Con respecto a la morfología y propiedades de los puntos debemos notar la evolución, pasando de formas circulares ([6], [28]), a puntos generados estocásticamente con otras formas más irregulares ([29], [2]) y a puntos generados mediante empaquetamiento espectral ([5]).

Entre los trabajos más relacionados con este proyecto, debemos destacar el trabajo de Isenberg et al. ([12]) centrado en comprender las valoraciones y diferencias percibidas por un ser humano al observar puntos generados por computador y puntos creados a mano; el trabajo de Arroyo et al. ([2]) que define un algoritmo estocástico capaz de generar puntos simulando el tamaño de la pluma, la viscosidad de la tinta y la capacidad de absorción del papel; y los trabajos de Martín et al. ([20], [21]), centrados en el desarrollo de un algoritmo de punteado eficiente mediante técnicas de halftoning, centrado en obtener la mayor calidad posible de impresión para comparar con ejemplos hechos a mano. Además recogen las técnicas usadas por los artistas al crear este tipo de obras, gracias a la colaboración con una artista local.

Finalmente, y para facilitar la comparación con los resultados obtenidos en este trabajo, se incluyen imágenes (figuras 3.3 y 3.4) con el proceso de edición manual de la entrada y el resultado obtenido por el equipo de Martín et al. [21].

En la figura 3.3 se observa el proceso de edición manual de la imagen de entrada, partiendo de una imagen en escala de grises (imagen de la izquierda), ajustando el contraste global y el detalle local (imagen central) y finalmente añadiendo los bordes, aplicando el recurso expresivo de la inversión y ajustando el contraste de forma local (imagen de la derecha). En la figura 3.4 se



Figura 3.3: Proceso de edición manual de la imagen de entrada (Fuente: [21]).

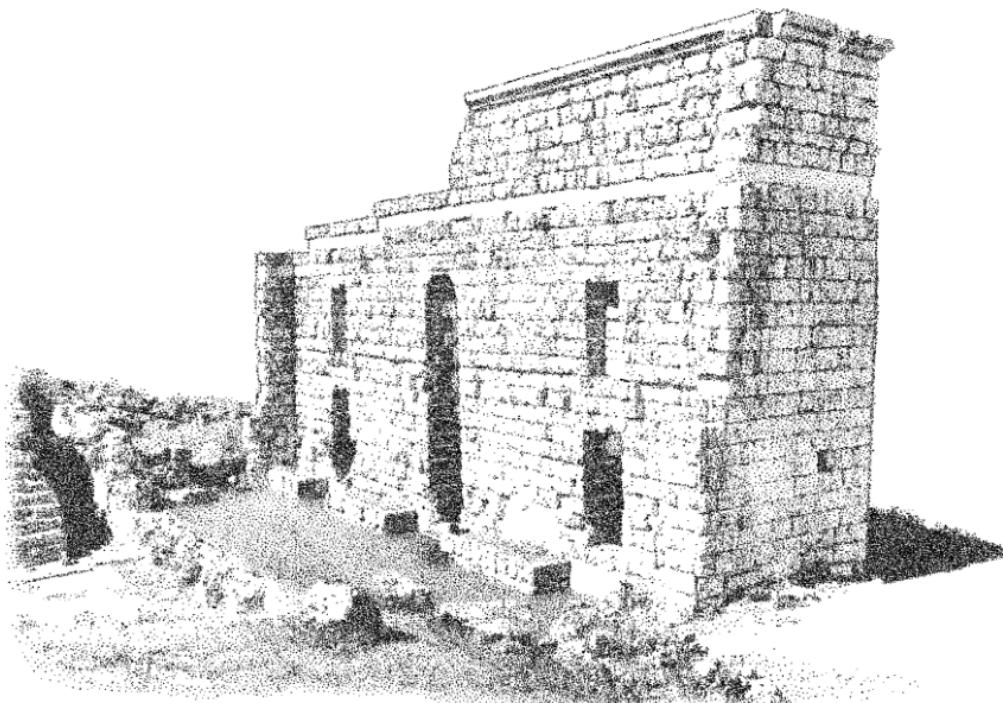


Figura 3.4: Imagen punteada resultante, generada para ser impresa en A4 a 1200dpi (Fuente: [21]).

observa el resultado de aplicar su algoritmo y orientando el procesamiento a obtener una imagen final imprimible en A4 a 1200dpi.

Capítulo 4

Descripción de Stippling

Stippling es el término inglés que designa el proceso de creación artístico conocido en español como Punteado. Dicho proceso consiste en la creación de patrones que simulen distintos grados de solidez mediante el uso de pequeños puntos.

Este tipo de patrones aparecen a menudo en la naturaleza, y son frecuentemente emulados por los artistas.

Esta técnica es usada frecuentemente y es, por tanto, natural que se intente simular por computador.

Este proyecto pretende llegar más lejos, intentando mejorar el algoritmo mediante el aprovechamiento de información tridimensional del objeto a puntear, para así poder reducir o evitar la necesidad de preprocesamiento manual de la imagen a puntear.

Los puntos generados usando la técnica descrita por Arroyo et al. ([2]), han demostrado ser creíbles, y pasar por auténticos, contando con la ventaja añadida de poder evitar el engorro de tener que escanear una cantidad suficientemente grande y variada de puntos reales, y su posterior acumulación de forma ordenada para su aprovechamiento. Tanto los puntos escaneados como los puntos generados son ampliamente mejores que los puntos creados mediante modelos. Es por esta razón que se han usado puntos generados estocásticamente en el proyecto, si bien no se ha implementado el algoritmo, ya que los puntos resultantes, almacenados en una imagen, son muy fácilmente aprovechables.

El algoritmo parte de una imagen, que es procesada, y genera los puntos de la imagen final punteada. Debido a la exigencia de entrada de los algo-

ritmos de tramado, el primer paso es pasar la imagen a escala de grises. A continuación, se calcula una imagen tramada a partir de la imagen en escala de grises. Para ello, Martin et al. proponen usar el método de Floyd-Steinberg por su celeridad y resultados razonablemente buenos, sin embargo, en el proyecto se ha incluido también el método de Stucki debido a que produce una imagen tramada resultante de mayor calidad, sin perjudicar demasiado la eficiencia.



Figura 4.1: Imagen punteada usando el método de Floyd-Steinberg



Figura 4.2: Imagen punteada usando el método de Stucki

En las figuras 4.1 y 4.2 se observa una comparación de imágenes punteadas a usando ambos algoritmos, punteando un cien por cien de los píxeles activos en la imagen tramada, con un factor de empaquetamiento igual a dos y sin realizar ninguna perturbación estocástica de la posición de los puntos. Tal como se explica con mayor detalle en el apéndice C, el método de Stucki produce unos resultados más precisos y siluetas más definidas, lo que influye mucho en la calidad de la imagen punteada resultante.

El primer paso, es, usando la imagen tramada (imagen binaria), para cada pixel activo, generar un punto de stippling. Hay que tener en cuenta que al generar los puntos de stippling de esta forma, no se obtiene una imagen punteada creible, ya que hay una correspondencia directa entre la imagen

tramada y la imagen punteada. Para solucionarlo, se recurre al factor de empaquetamiento, cuyo objetivo es controlar la densidad de puntos de la imagen punteada final para así hacerla más realista y parecida a un trabajo hecho a mano por un artista.

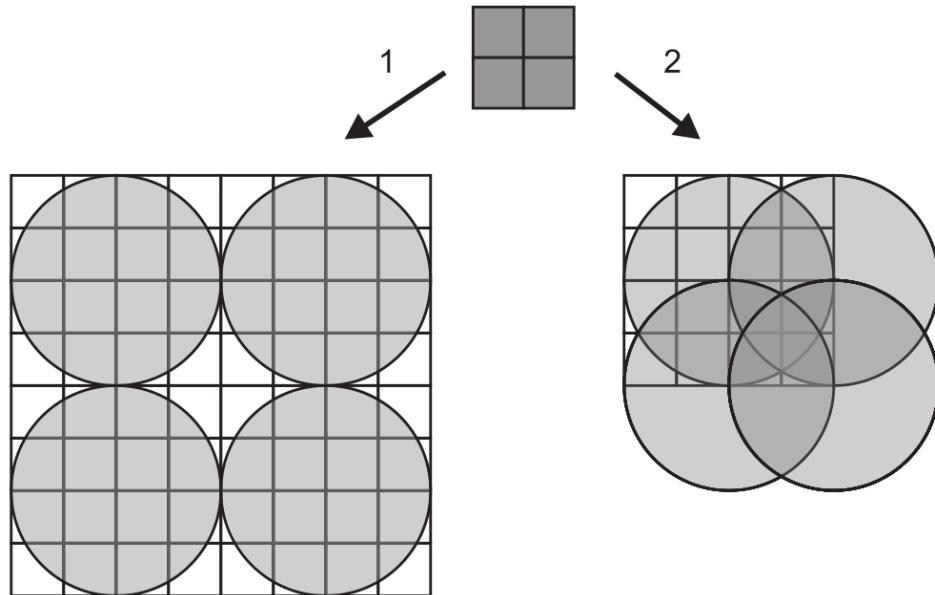


Figura 4.3: Efecto del factor de empaquetamiento (Fuente: [21])

Por supuesto, el artista cuando trabaja a mano, tiene la posibilidad de escoger el tamaño de la pluma que utiliza para puntear, atendiendo a sus propios criterios y recursos expresivos. Para simular este hecho, en la aplicación, se deberá permitir al usuario escoger el conjunto de puntos, indicando una resolución en dpi y la pluma a la que son análogos. Por supuesto, el escoger un grupo de puntos u otro, será determinante en la resolución de la imagen punteada final, ya que es directamente proporcional al tamaño de dichos puntos.

$$Fw = \frac{Dw * Sw}{Pf} \quad (4.1)$$

Figura 4.4: Fórmula para calcular el ancho de la fotografía punteada final.

$$Fh = \frac{Dh * Sh}{Pf} \quad (4.2)$$

Figura 4.5: Fórmula para calcular el alto de la fotografía punteada final.

Las fórmulas 4.4 y 4.5 permiten determinar el tamaño de la imagen punteada a generar. En ellas, Dw y Dh representan el ancho y el alto respectivamente de la imagen trama (dithered); Sw y Sh representan el ancho y el alto de los puntos (stipples) usados al puntear y Pf representa el factor de empaquetamiento. Además, se puede reducir el tamaño de la imagen de entrada (antes de aplicar el algoritmo de trama) para ajustar la imagen final a una superficie específica de impresión, al permitir al usuario escoger la resolución en ppp (puntos por pulgada, en inglés dpi, dots per inch) de los puntos de punteado (stipples). Para ello, no hay más que incluir el factor de reducción en el divisor de la fórmula anterior.

Una de las mejoras indicadas en el artículo consiste en la perturbación estocástica de la posición de los puntos generados. Debido a que los puntos tienen una correspondencia directa con los píxeles activos de la imagen trama, el resultado es demasiado regular, dando la impresión de que los puntos están encuadrados en una rejilla. Es por esto que se aplica dicha perturbación, para lograr una distribución más realista de los puntos.

Otra de las innovaciones no contenidas en el artículo consiste en el uso de información tridimensional que represente los objetos fotografiados para mejorar la imagen punteada resultante. Normalmente las imágenes de entrada son preparadas a mano, eliminando zonas que no se desean puntear, o editando otras para lograr un mejor resultado. En este caso, crear un modelo tridimensional de los objetos a puntear permite un mayor control del resultado final.

Para entender mejor cómo funciona, se procede a detallar los dos tipos de configuración que se permiten en el algoritmo, una configuración general, y configuraciones específicas asociadas a modelos o partes de un modelo concretos.

La configuración general permite controlar las cantidades de puntos que se generan en zonas pertenecientes y no pertenecientes al modelo. Las configuraciones específicas permiten controlar la cantidad de puntos que se generan en zonas pertenecientes a un modelo concreto, así como la cantidad de puntos

pertenecientes a la silueta de dicho modelo que pueden sufrir perturbaciones de posición (atendiendo a la configuración general, las siluetas no se perturban, para así mantener las líneas que las definen). De esta forma, creando y configurando cuidadosamente el modelo, se puede obtener un resultado final que permite puentear unas zonas con mucho detalle, y otras con menos, así como la calidad del mismo.

Capítulo 5

Planificación.

5.1. Planificación inicial

El proyecto ha seguido desde el principio una metodología ágil tipo Scrum ([27]). Aunque no se ha seguido rigurosamente, ya que durante casi todo el proyecto (excepto las contribuciones que Germán hizo al proyecto y que están comentadas en el apéndice D) he sido el único desarrollador, lo que significaba que todas las historias de usuario me correspondían a mí, y que el sprint debía incluir tanta funcionalidad como fuese posible, respetando las prioridades del cliente/usuario, en función de mis capacidades.

El lema de las metodologías ágiles es "welcome the changes", que contrasta con la rigidez de las metodologías tradicionales herederas de RUP. De hecho, la planificación que se realiza es muy abstracta, y sin asignar tareas específicas a rangos de tiempo o a personas, ya que es en las reuniones donde el cliente/usuario y el equipo de desarrollo discuten las prestaciones necesarias y se plasman en historias de usuario. También en esas reuniones se decide la temporización del sprint de desarrollo, pues para sprints con muchas historias, o muy complejas, la duración de la iteración puede y debe variar.

Al comienzo del proyecto se realizó una estimación de la duración del desarrollo del proyecto y se establecieron los hitos más relevantes. Debido a que el proyecto se iría desarrollando usando una metodología ágil, y podría sufrir cambios posteriores, no se realizó una planificación más exhaustiva. La dinámica de trabajo incluiría reuniones semanales y, como el equipo estaría integrado por un solo miembro, las historias de usuario se mantendrían en apuntes sobre papel en lugar de sobre un mural.

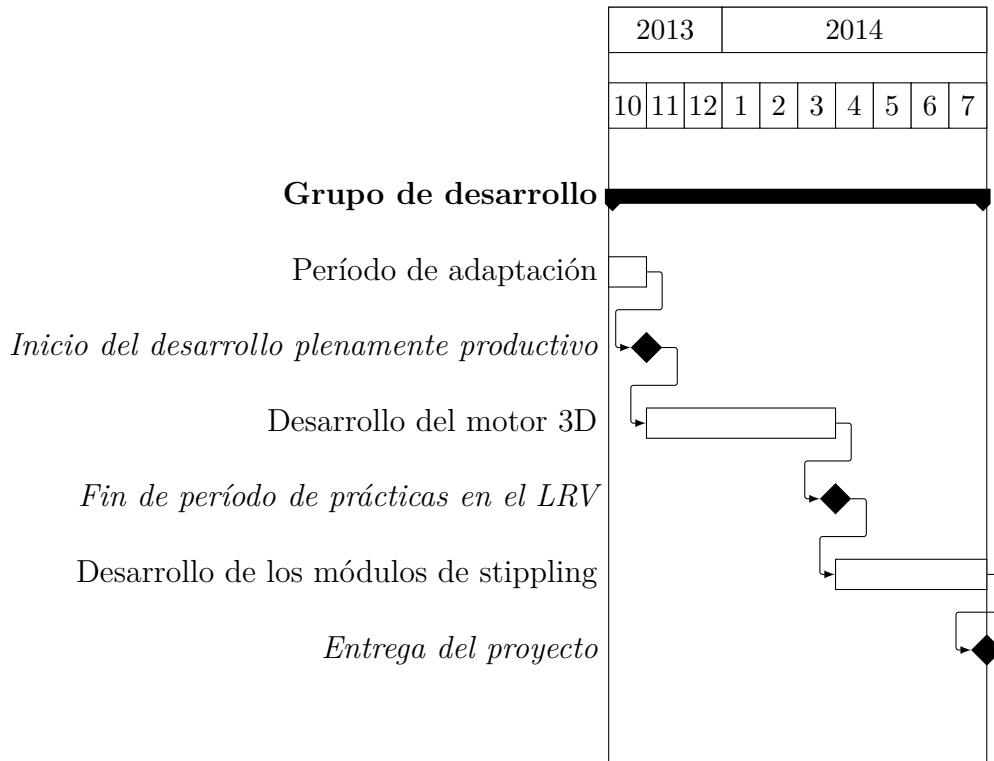


Figura 5.1: Planificación inicial.

El período de adaptación consistiría en aproximadamente un mes de adaptación a las tecnologías y estándares de calidad del Laboratorio de Realidad Virtual de la Universidad de Granada, además de la integración con el resto de los miembros del laboratorio. Tal cantidad de tiempo sería necesaria debido a que en el LRV se usan las últimas versiones de las tecnologías disponibles, y desgraciadamente, necesitaría actualizar mis conocimientos, especialmente los relacionados con las últimas versiones de OpenGL y GLSL (GL Shading Language).

El período de desarrollo del motor tridimensional, incluye los dos primeros objetivos mencionados en la sección de objetivos (Reconstrucción de escena y desarrollo de un Motor 3D con capacidad de creación de modelos complejos e iluminación). Este período estaría incluiría las prácticas de empresa realizadas en el Laboratorio de Realidad Virtual, y sería el más intensivo, tanto en horas, como en carga de trabajo, debido a que se desarrollaría toda la arquitectura del proyecto a la vez que se diseñarían e implementarían específicamente los objetivos mencionados.

Finalmente, el período de desarrollo de los módulos de Stippling, se centraría en completar el último de los objetivos definidos. En este período, las prácticas de empresa en el Laboratorio de Realidad Virtual habrán finalizado, pero no así la colaboración con el equipo del LRV, que continuará apoyando el proyecto hasta su finalización. En esta parte se deberá desarrollar el algoritmo correspondiente al último artículo de investigación publicado en este campo por el LRV [21], y con el apoyo del equipo, especialmente de Germán Arroyo, mejorarlo, haciendo uso de las posibilidades que provee el motor tridimensional diseñado a tal efecto.

5.2. Desarrollo del proyecto e incidencias

En general el proyecto se ha ido desarrollando acorde a la planificación inicial. Aunque han ocurrido algunos retrasos, siempre se ha podido recuperar el ritmo sin que afectasen al proyecto.

Octubre-Noviembre

El primer hito se alcanzó una semana antes de lo esperado gracias a que el conjunto de ejemplos que Germán me encargó fueron inteligentemente diseñados para acelerar el desarrollo del proyecto. El periodo formativo (especialmente referente a GLSL) continuó durante unas dos semanas más hasta conseguir un dominio lo bastante bueno del desarrollo de shaders.

Diciembre-Enero

Mas tarde en Diciembre-Enero ocurrió un pequeño retraso (más o menos una semana) relacionado con la integración de la biblioteca CGAL (biblioteca destinada al desarrollo de modelos mediante geometría constructiva) que se arrastró hasta febrero, momento en que de nuevo se alcanzó el progreso asignado en la planificación inicial. Durante ese periodo también se solucionó un problema de integración del modelo de árbol de geometría constructiva con Qt.

Enero-Marzo

Tras esto, y hasta que se alcanzó el segundo hito se desarrolló la vista ló-

gica del árbol CSG, el sistema de renderización de entidades como sólidos iluminados. Aunque se alcanzó el segundo hito con un par de semanas de retraso, debido a que se tuvo que adaptar el diseño de forma que soportase el renderizado mediante el cauce antiguo (para la iluminación), dicho retraso no afectó al desarrollo del proyecto.

Marzo-Julio

Una vez alcanzado el segundo hito, se celebraron varias reuniones en que Germán concretó los detalles de la implementación de la parte de stippling. Esta parte, gracias al sólido diseño realizado durante la primera parte, pudo ser implementada con mayor celeridad, y el mayor problema que apareció, fue la falta de precisión que la edición de sólidos mediante una sola vista inducía. Este problema se solucionó creando una segunda vista superior para permitir al usuario visualizar con precisión la escena. También se añadieron una serie de requerimientos nuevos, con el objetivo de pulir la aplicación para la presentación, basados en permitir al usuario establecer configuraciones de punteado específicas a cada parte del árbol CSG. Las dos últimas semanas se dedicaron a crear la documentación y a ultimar detalles del proyecto.

Capítulo 6

Análisis

A continuación se presenta una recopilación de las historias de usuario que se han ido recibiendo durante el desarrollo del proyecto. Es importante observar que no todas han llegado a implementarse o han llegado a la etapa final de desarrollo sin sufrir cambios, ya sea por necesidades técnicas o por cambio en las necesidades del usuario final. Algunas representan requerimientos de alto nivel, y otras representan requerimientos técnicos específicos, propios de un proyecto de investigación. En general, todos se han redactado de la forma más abstracta y general posible, para hacer de esta sección descriptiva pero breve.

- Crear una aplicación que contenga un visor de OpenGL.
- La aplicación deberá mostrar líneas editables de fuga que permitan al usuario definir la escena.
- La aplicación deberá poder representar figuras geométricas tridimensionales básicas (prismas, cilindros, prismas de base poligonal, prismas de base curvilínea...) mediante el uso de puntos de control.
- La aplicación deberá permitir la interacción con los sólidos para definir su geometría, posición y orientación.
- La aplicación deberá usar un sistema de selección moderno y eficiente (se propone la selección usando colores como identificadores únicos).
- La aplicación deberá ser capaz de cargar y representar imágenes de fondo en el visor de OpenGL, y soportar interacciones típicas (desplazamiento y zoom).

- La aplicación deberá ser capaz de leer los metadatos de la imagen cargada, y en caso de no encontrarse el metadato de la distancia focal, dar la opción al usuario de especificar los datos necesarios.
- La aplicación deberá ser capaz de reconstruir la escena a partir de los metadatos extraídos de la imagen.
- La aplicación deberá proporcionar un menú que permita al usuario cambiar los parámetros de cámara(altura, ángulo de inclinación...).
- Las líneas de fuga se sustituyen por un plano que represente el suelo.
- La aplicación deberá ser capaz de extraer coordenadas 3D relacionadas con la posición del ratón (fijando un plano xy o xz).
- La aplicación deberá poder representar los sólidos como una rejilla (wireframe), además, se plantea la posibilidad de que sobre las caras de los sólidos se puedan renderizar partes de la imagen como texturas.
- La aplicación deberá poder obtener nuevos sólidos complejos a partir de otros creados por el usuario mediante operaciones booleanas.
- La aplicación deberá mostrar un indicador visual que permita distinguir la altura a la que se encuentra el sólido sobre el suelo.
- La aplicación deberá proveer una pantalla de configuración que permita cambiar los colores de los múltiples elementos renderizados (puntos de control, sólidos, suelo...).
- La aplicación deberá incluir una vista de árbol que permita interacciones para definir el árbol CSG. Dicha vista de árbol deberá incluir opciones para modificar la visibilidad del sólido, su nombre, la visibilidad de sus puntos de control y, en caso de ser una operación, el tipo de operación.
- Se descarta la renderización de texturas sobre los sólidos en favor de renderizaciones como sólidos mate y sólidos iluminados.
- La aplicación deberá poder eliminar sólidos del árbol CSG.
- La aplicación deberá ser capaz de guardar y cargar el árbol CSG.
- La aplicación deberá ser capaz de cargar y procesar imágenes que contengan matrices de puntos (stipples).

- La aplicación deberá ser capaz de obtener una imagen punteada básica (siguiendo el algoritmo de stippling descrito).
- La aplicación deberá permitir definir la semilla de las distribuciones de números pseudoaleatorios para poder obtener resultados reproducibles.
- La aplicación deberá permitir al usuario definir y aplicar una perturbación estocástica a la posición de los puntos.
- La aplicación deberá ser capaz de guardar y cargar los puntos generados para la imagen punteada.
- La aplicación deberá ser capaz de realizar un análisis de bordes (método de Sobel) sobre los modelos creados, de forma que dichos bordes se usen para evitar que los puntos (stipples) asociados sufran perturbaciones estocásticas de sus posiciones.
- La aplicación deberá permitir definir opciones generales de punteado de forma que el usuario pueda definir la cantidad de puntos que se generan (pertenecientes, y no pertenecientes al modelo tridimensional).
- La aplicación deberá permitir definir el factor de empaquetamiento y aplicarlo sobre la imagen punteada.
- La aplicación deberá permitir al usuario cambiar entre los modos de renderización en textura única o en múltiples texturas (requerido con el objetivo de comparar eficiencia), además de definir el número de texturas en el modo de múltiples texturas.
- La aplicación deberá incluir visualizaciones de apoyo: vista superior y vista esquemática de escena (vista en que se muestra un esquema que representa la posición de la cámara, el ángulo de visión, el suelo y la posición estimada del plano objeto).
- La aplicación deberá incluir, además del método de detección de bordes de Sobel, el método de Canny, y permitir al usuario escoger el método a usar.
- La aplicación deberá permitir definir opciones de punteado específicas (referentes a un modelo específico), de forma que el usuario pueda decidir la cantidad de puntos de la silueta que sufren la perturbación estocástica de posición (por defecto a cero), la cantidad de puntos internos al modelo que se generan, y el método de detección de bordes.

- La aplicación deberá dar al usuario la opción de escoger el conjunto de puntos a usar (indicando su ppp y el tamaño de la pluma que representan) e indicar una vez se ha generado la imagen final, la superficie óptima de impresión y su ppp.
- Añadir un modificador de la distancia de cámara para permitir al usuario un mayor control.

Capítulo 7

Diseño

7.1. Diseño estructural

A continuación se detallará el diseño del proyecto, empezando desde un punto de vista abstracto, y poco a poco concretando los módulos. El proyecto posee dos módulos productivos claramente diferenciados, el motor tridimensional usado para desarrollar los modelos 3D, y el motor de Stippling, usado para calcular, generar y renderizar los puntos de la imagen punteada. El resto de componentes corresponden a la interfaz gráfica o a la estructura de clases de soporte (sólidos, operaciones CSG, QuadTree, etc.). Por supuesto, sendos motores tienen un componente intrínseco de interfaz gráfica pues son visores de OpenGL.

Como se puede observar en la figura 7.1, la interfaz gráfica está integrada por la clase MainWindow, cuyos componentes principales son los motores mencionados (clases GLWidget3DEngine y GLWidgetStippling), la vista del árbol CSG (clase EntityTreeWidget), la vista de apoyo superior (clase GLWidget3DEngineSuperiorVisualization), entre otros componentes de menor importancia (control de cámara, diálogos de configuración, etc.).

El diseño del motor 3D ha seguido un enfoque MVC (Model-View-Controller). En la figura 7.2 se pueden observar los componentes relacionados con el motor 3D, colocando a la izquierda las vistas (clases GLWidget3DEngine, GLWidget3DEngineSuperiorVisualization y EntityTreeWidget), en el centro el controlador (EntityTreeController) y a la derecha las clases referentes al modelo (EntityTreeNode y Entity3D). Debido a su especial relevancia en el motor tridimensional, la vista del árbol CSG se ha incluido aquí, y se detallará más

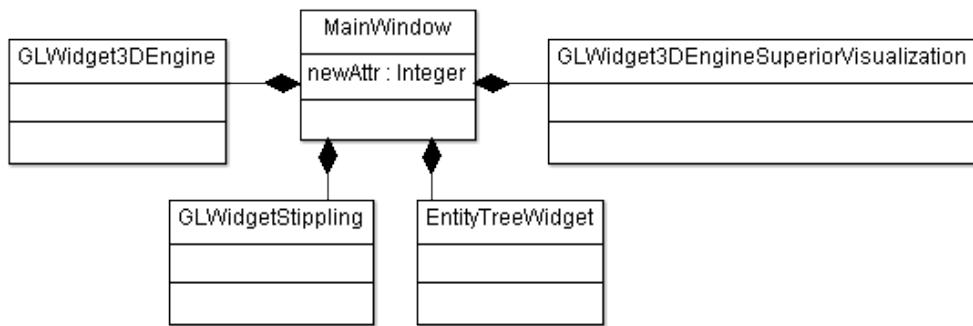


Figura 7.1: Componentes principales de la interfaz gráfica del proyecto

adelante.

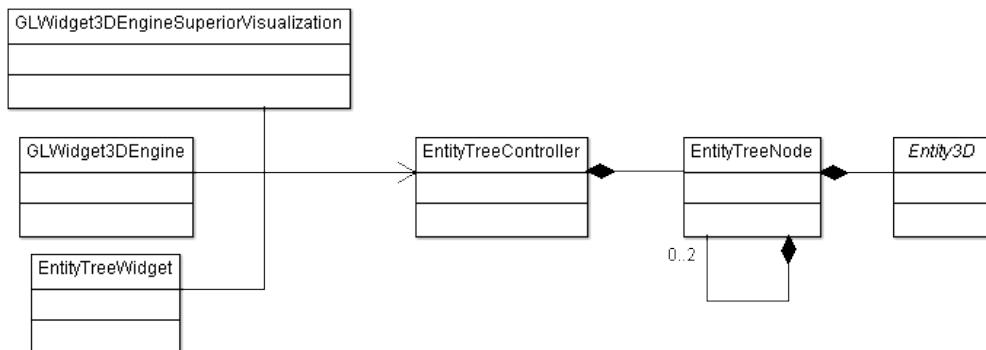


Figura 7.2: Componentes relacionados con el motor 3D organizados según el modelo MVC

Al igual que con el motor 3D, el motor de Stippling sigue un enfoque MVC, como se puede observar en la figura 7.3. De nuevo en el lado izquierdo aparece la vista (GLWidgetStippling), en el centro el controlador (QuadTree) y a la derecha, los componentes del modelo (QuadTreeNode y StippleDot). Es importante destacar el papel de la clase DotGenerationWorker, que se comporta como un productor asíncrono al servicio de la interfaz, pues se encarga de generar el QuadTree y los stipple para garantizar la interactividad de la interfaz.

A continuación se detallarán los componentes del árbol CSG, concretando posteriormente aún más su componente más básico, las entidades 3D.

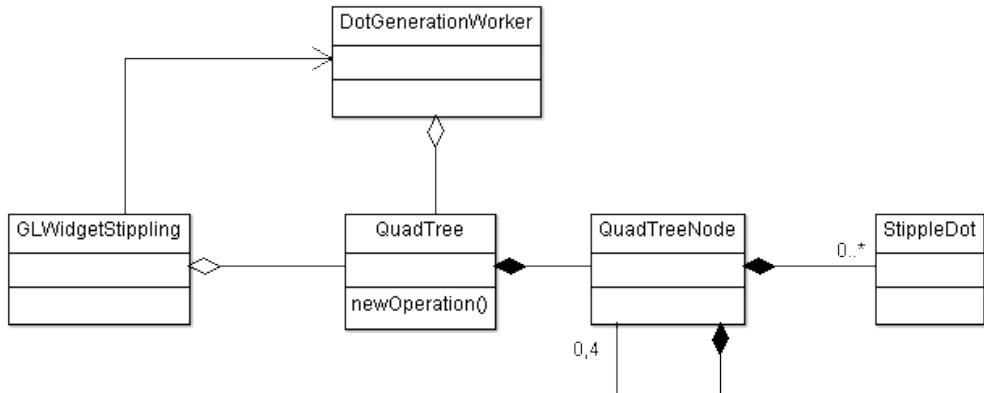


Figura 7.3: Componentes relacionados con el motor de stippling organizados según el modelo MVC

Como podemos observar en la figura 7.4, el árbol CSG tiene componentes visuales y componentes lógicos. Es importante notar que también aquí se respeta el enfoque MVC. La clase **EntityTreeWidgetItem** representa un nodo del árbol, y constituye la representación visual del nodo **EntityTreeNode**. Por supuesto, la vista tiene el control sobre los mismos, y los crea, modifica o destruye atendiendo a la información que el controlador (**EntityTreeController**) proporciona.

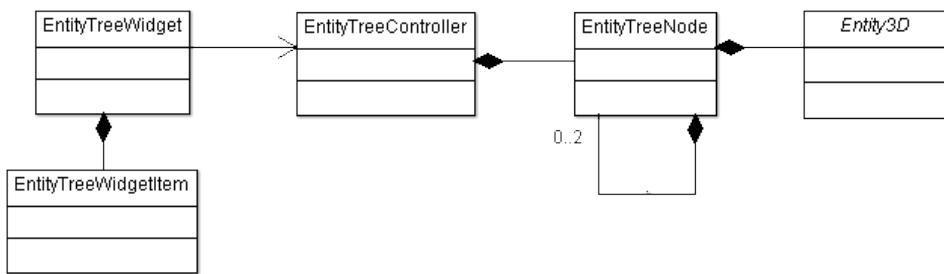


Figura 7.4: Componentes del árbol CSG

La clase **EntityTreeNode** es parte del modelo del árbol CSG, y representa un nodo. Posee una relación reflexiva que modela su recurrencia (un nodo tiene nodos descendientes). Cada nodo tiene una entidad 3d correspondiente

(Entity3D), que representa al sólido de dicho nodo.

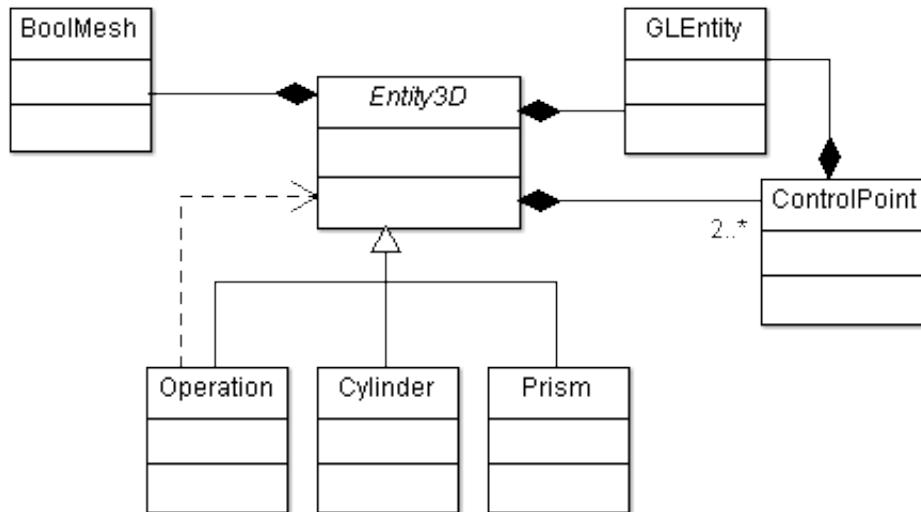


Figura 7.5: Componentes que definen una Entidad 3D

En la figura 7.5, podemos observar los componentes que definen una entidad 3d (Entity3D). La generalización permite especificar una geometría a la entidad 3d (puesto que la entidad 3d es una clase virtual que debe especificarse para ser útil) mediante una serie de puntos de control (ControlPoint). La clase BoolMesh, es el componente que realiza los cálculos correspondientes a las operaciones booleanas entre sólidos, y cada Entity3D posee un BoolMesh que lo define. Las operaciones (Operation) usan los BoolMesh de sus operandos para calcular sus propios BoolMesh (que definirán la geometría, junto a los puntos de control y que será renderizada por la GLEntity).

Finalmente, es importante notar que el conjunto de componentes que definen los sólidos están entre los componentes más concretos del diseño, en particular la clase GLEntity, que representa una entidad representable en OpenGL (un punto, una superficie, un volumen; admite la aplicación de texturas y el uso de distintos shader, es seleccionable mediante el método de colores como id... En definitiva, la clase GLEntity es flexible y es la base de los procesos de renderizado en la aplicación.

Por supuesto, el proyecto contiene muchas más clases que no se mencionan

aquí, cuyo propósito es de apoyo en la definición de la estructura mencionada, y cuya concrección haría de este diseño demasiado extenso y complicado de entender. Entre las más relevantes, cabe mencionar la clase IDManager, que representa un controlador de identidades únicas, la clase GLCamera, que representa una cámara de OpenGL (permite traslaciones, rotaciones y genera la matriz de vista, como si de un objeto más se tratase), o las clases HeightIndicator y FloorPlane (sendas clases destinadas a servir de apoyo en la visualización de los sólidos).

7.2. Diseño de interacciones

Las interacciones con sólidos mediante puntos de control, tal como fueron descritas por Germán Arroyo, fueron modeladas como storyboards a mano, como apoyo para su correcta implementación, y han sido digitalizadas mediante la propia aplicación para ayudar al lector a familiarizarse con la interfaz.

7.2.1. Posicionamiento de un modelo

En las figuras 7.6, 7.7 y 7.8 se observa la interacción de posicionamiento de un modelo, en este caso, de un cilindro. En la figura 7.6, se haría click sobre el punto de control rojo, y sin soltar, se arrastraría el ratón, observando como se modifica la posición del cilindro con el movimiento del puntero del ratón. En el momento en que se esté satisfecho con la posición del cilindro, no hay mas que soltar el botón del ratón, como se observa en la figura 7.7.

Por supuesto, además de poder mover el cilindro sobre un plano horizontal, se puede también mover sobre un plano vertical. Para ello no hay más que repetir la misma interacción que se ha descrito antes, sólo que manteniendo la tecla *Ctrl* pulsada durante todo el proceso de la interacción. El resultado final se puede observar en la figura 7.8, así como también se observa bajo el modelo el indicador visual de altura.

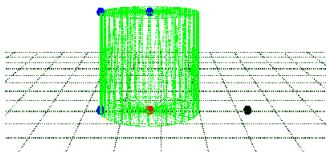


Figura 7.6: Modelo original.

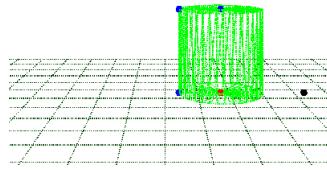


Figura 7.7: Modelo tras ser reposicionado sobre el plano horizontal sobre el que se sitúa.

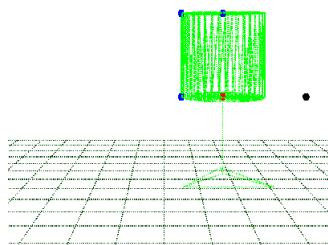


Figura 7.8: Modelo tras ser reposicionado sobre el plano vertical sobre el que se sitúa.

7.2.2. Rotación de un modelo en torno al eje Y

En las figuras 7.9, 7.10 y 7.11 se observa la interacción de rotación en torno al eje Y de un modelo, en este caso, de un prisma. En la figura 7.9, se haría click sobre el punto de control negro que se sitúa en el lado derecho de la base del prisma, y sin soltar, se arrastraría el ratón alrededor del punto de posición rojo, observando como se modifica el prisma con el movimiento del puntero del ratón. En el momento en que se esté satisfecho con el ángulo del modelo, no hay mas que soltar el botón del ratón, como se observa en las figuras 7.10 y 7.11.

7.2.3. Redimensionado de altura

En las figuras 7.12 y 7.13 se observa la interacción de cambio de altura de un modelo, en este caso, de un cilindro. En la figura 7.12, se haría click sobre el punto de control azul que se sitúa en el centro de la base superior del cilindro, y sin soltar, se arrastraría el ratón, observando como se modifica el cilindro con el movimiento del puntero del ratón. En el momento en que se esté satisfecho con la altura del cilindro, no hay mas que soltar el botón del ratón, como se observa en la figura 7.13.

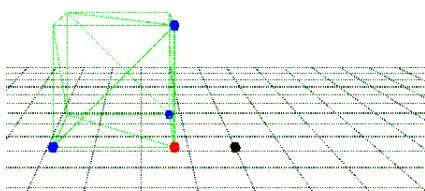


Figura 7.9: Modelo original.

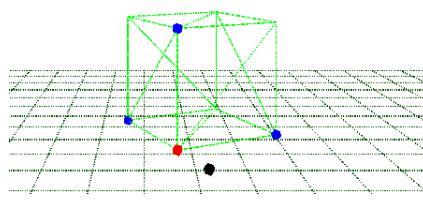


Figura 7.10: Modelo experimentar una rotación.

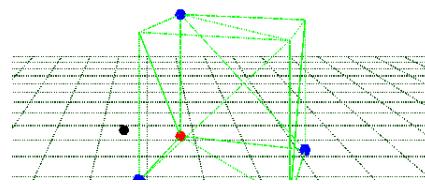


Figura 7.11: Modelo experimentar una subsiguiente rotación.

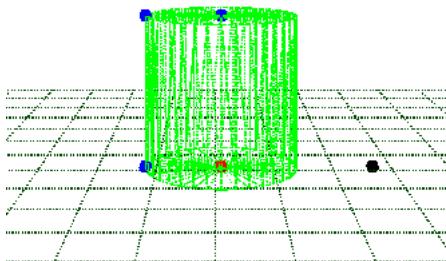


Figura 7.12: Modelo original.

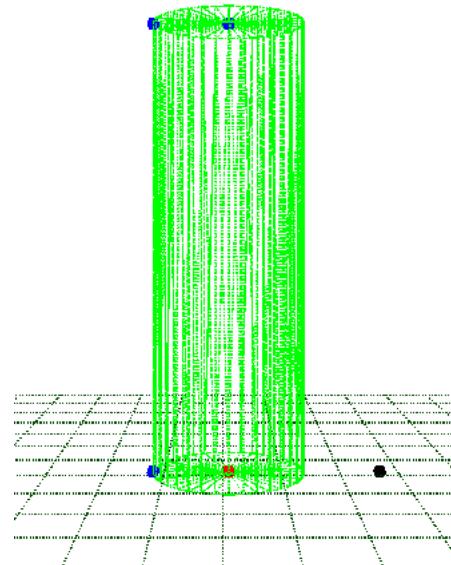


Figura 7.13: Modelo tras redimensionar su radio inferior.

7.2.4. Redimensionado del radio

En las figuras 7.12 y 7.13 se observa la interacción de cambio de radio de un modelo, en este caso, de un cilindro. En la figura 7.12, se haría click sobre el punto de control azul que se sitúa en el radio de la base inferior del cilindro, y sin soltar, se arrastraría el ratón, observando como se modifica el cilindro con el movimiento del puntero del ratón. En el momento en que se esté satisfecho con la altura del cilindro, no hay mas que soltar el botón del ratón, como se observa en la figura 7.13.

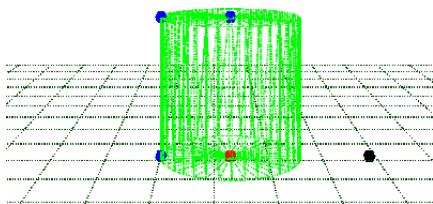


Figura 7.14: Modelo original.

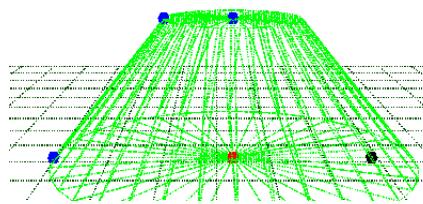


Figura 7.15: Modelo tras redimensionar su radio inferior.

7.2.5. Redimensionado de caras planas

En las figuras 7.16, 7.17 y 7.18 se observa la interacción de redimensión de caras planas de un modelo, en este caso, de un prisma. En la figura 7.16, se haría click sobre el punto de control azul que se sitúa en el lado frontal izquierdo de la base del prisma, y sin soltar, se arrastraría el ratón, observando como se modifica el prisma con el movimiento del puntero del ratón. En el momento en que se esté satisfecho con el tamaño de la cara del prisma, no hay mas que soltar el botón del ratón, como se observa en la figura 7.13.

Ya que el prisma puede definir sus caras aprovechando la simetría, sólo se pueden definir los tamaños de dos de sus caras, y como se observa en la figura 7.18, se pueden modificar ambas caras siguiendo la misma forma de proceder que se ha indicado.

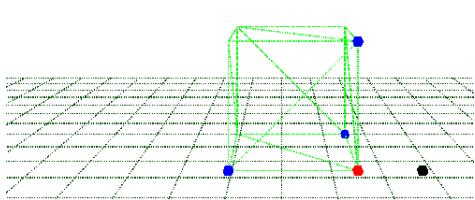


Figura 7.16: Modelo original.

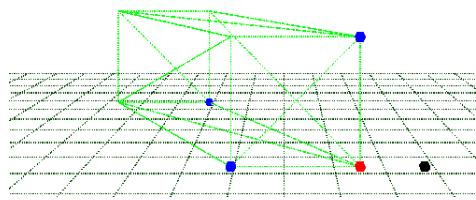


Figura 7.17: Modelo tras redimensionar una de sus caras.

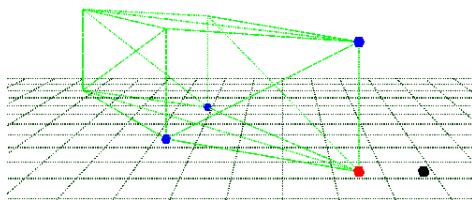


Figura 7.18: Modelo tras redimensionar otra de sus caras.

Capítulo 8

Algoritmos usados

8.1. Reconstrucción de escena a partir de metadatos de una fotografía

Uno de los retos que presentaba este proyecto, era el de la reconstrucción de la escena a partir de los metadatos presentes en el fichero de la imagen proporcionada para ser punteada. Debido a que las fotografías modificadas y a que no todas las cámaras escriben estos metadatos, en caso de no ser encontrados, se pide al usuario que los especifique, por lo que en general, se puede suponer que siempre estarán disponibles.

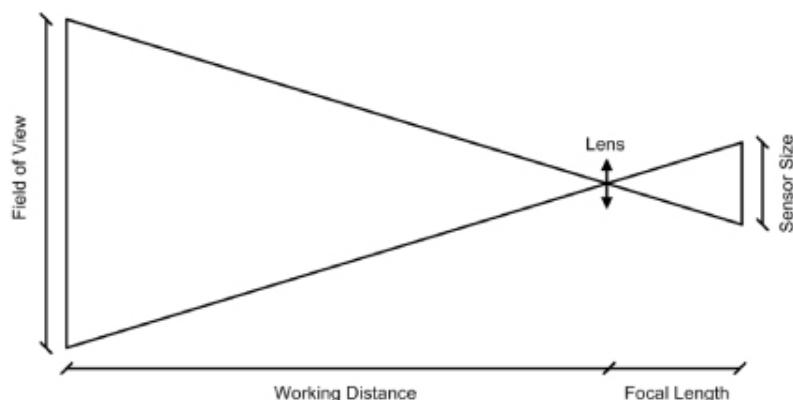


Figura 8.1: Esquema de proyección en una cámara digital (Fuente: [13])

Por reconstrucción automática de la escena no debe entenderse reconstrucción tridimensional completa de la misma, esto es imposible a partir de una sola fotografía. Para reconstruir la escena, es necesario conocer el ángulo

de campo de visión de que disponía la cámara en el momento de tomar la fotografía. Para ello, nos servimos del modelo de proyección de las cámaras digitales (ver figura 8.1) donde podemos observar que el ángulo de visión es igual en el interior y en el exterior de la cámara por ser ángulos opuestos. Dado que no podemos conocer el tamaño de los objetos fotografiados ni la distancia a la que se encuentran, usamos el tamaño del sensor de la cámara, que obtenemos de la base de datos mediante el metadato que indica el modelo de la cámara. La distancia focal también se guarda como un metadato en la fotografía. De esta forma podemos calcular el VFOV (Vertical Field of View) que es el que OpenGL utiliza para proyectar.

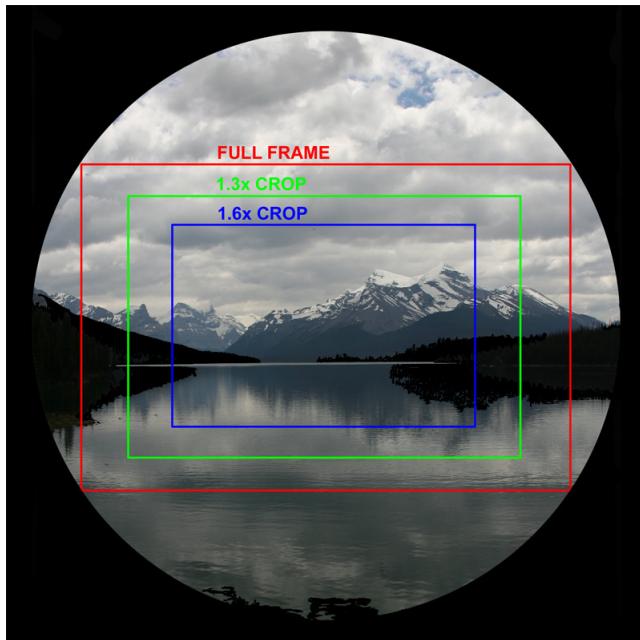


Figura 8.2: Comparación del efecto de crop factor sobre una fotografía (Fuente: www.talkphotography.co.uk)

Sin embargo, esto no basta, sino que hay que tener en cuenta el factor de recorte (crop factor en inglés) que las cámaras digitales actuales implementan. El crop factor también se conoce en español como factor de multiplicación de la distancia focal. Este nombre describe de forma literal su efecto sobre la distancia focal, y por tanto, sobre la fórmula del campo de visión.

A continuación se incluye la fórmula utilizada para calcular el campo de visión:

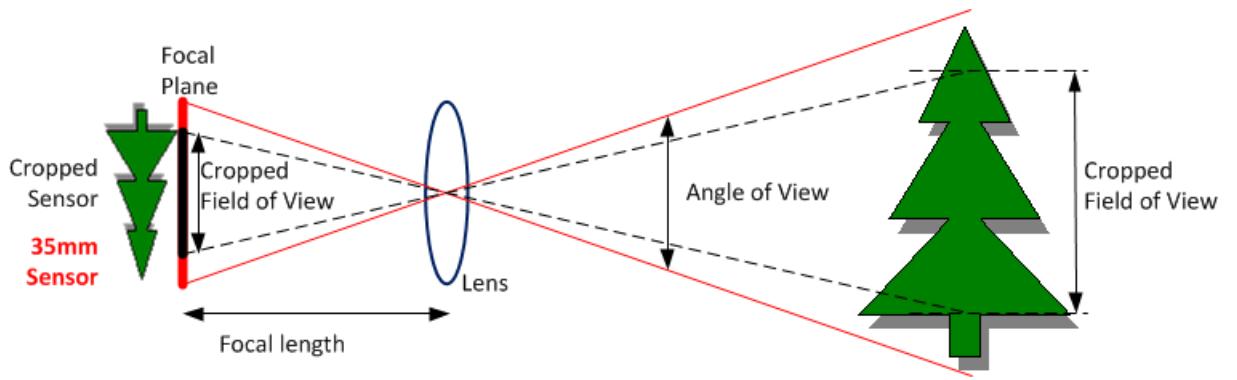


Figura 8.3: Efecto del crop factor en el modelo de proyección de una cámara digital (Fuente: Wikipedia)

$$VFOV = 2 * \arctan\left(\frac{Ch}{Fl * Cf}\right) * \frac{180}{\pi} \quad (8.1)$$

Figura 8.4: Fórmula para calcular el campo de visión en grados.

En dicha fórmula, VFOV representa el campo de visión vertical en grados, Ch representa la altura del sensor de la cámara (obtenido de la base de datos de cámaras), Fl representa la distancia focal (tomada a partir de los metadatos) y Cf representa el factor de recorte o crop factor (obtenido de la base de datos de cámaras).

$$Cd = \frac{\frac{Ih}{2}}{\tan\left(\frac{VFOV}{2}\right)}; \quad (8.2)$$

Figura 8.5: Fórmula para estimar la distancia de cámara.

Además, se puede estimar la distancia de cámara mediante la fórmula 8.5 suponiendo que la cámara está situada de forma perpendicular al plano objetivo y a una altura de la mitad de la altura de este.

8.2. Organización de puntos mediante un QuadTree

Un QuadTree es una estructura de datos integrada por un árbol en que cada nodo tiene cuatro hijos (o cero si es una hoja). Cada nodo representa un área de una superficie bidimensional. El QuadTree organiza el área de forma que cada hijo recibe un cuadrante, que a su vez reparte entre su prole. De esta manera, se puede especificar una zona concreta de la superficie manejada tan sólo estableciendo un código formado por dos bits por nivel de profundidad. A este código se le denomina QuadCode.

Además, ya que en cada nivel de profundidad el área se divide en cuatro, la eficiencia de acceso es logarítmica de base cuatro, lo cual es muy conveniente cuando se tiene gran cantidad de datos situados sobre la superficie manejada.

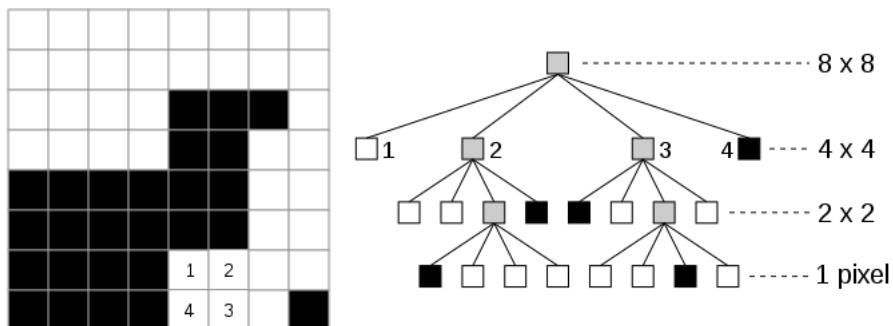


Figura 8.6: Imagen de mapa de bits y el QuadTree que la representa (Fuente: Wikipedia)

En el caso de el proyecto, cada punto (stipple) tiene una posición, y por tanto, pertenece a una rama del QuadTree que maneja el espacio definido por el tamaño de la imagen punteada. Organizar los puntos mediante un QuadTree permite mantener el orden garantizando una eficiencia de acceso óptima. Obtener un área contenida en el QuadTree es tan sencillo como hacer que cada nodo devuelva los puntos del área resultante entre la intersección de el área requerida y el área del propio nodo.

Sin embargo, en la aplicación existe un pequeño problema, ya que suele pasar que si el punto está fuera del área, debido a que su tamaño no es despreciable, debería renderizarse también, aunque fuese parcialmente. Afortunadamente,

este problema se soluciona de forma sencilla añadiendo un marco al área requerida, de forma que se incluyan también los puntos afectados.

Capítulo 9

Recursos, resultados y pruebas

9.1. Recursos

Los recursos que se han dispuesto para el desarrollo de este proyecto han sido humildes, sin necesidad de inversiones extra.

Durante la etapa inicial, enmarcada en el programa de prácticas de empresa de la Universidad de Granada, se desarrolló parcialmente el proyecto en uno de los ordenadores del Laboratorio de Realidad Virtual, asignado al llegar. Sin embargo, la mayor parte del desarrollo y prueba se realizó en mi ordenador personal.

Las características técnicas del ordenador personal son las siguientes:

- AMD Phenom II X4 945 (4 procesadores) 3GHz
- 8 GB RAM
- nVidia GeForce GTX 550Ti (1GB GDDR5)
- Sistemas operativos Ubuntu 13.10 64b y Windows 7 Ultimate 64b

También se ha utilizado de un ordenador portátil (bastante antiguo) cuya tarjeta gráfica no soportaba los requerimientos mínimos del proyecto, y que por tanto, sólo se ha usado para realizar la documentación.

Es importante observar que, debido a los requerimientos mínimos del proyecto, la tarjeta gráfica debe soportar al menos OpenGL v3.3.

En cuanto al software usado durante el desarrollo, se ha intentado usar software libre en la medida de lo posible, exceptuando el caso del desarrollo para Windows, en que se ha usado Visual Studio por comodidad.

- Qt Creator 2.7.1
- CMake 2.8.11.2
- Visual Studio 2010
- TeXnicCenter 2.02

9.2. Resultados y pruebas

En esta sección se presentará una serie de resultados, empezando por un desarrollo de un modelo 3D de una imagen y su posterior punteado, para pasar después a comparar diversas opciones de punteado y capacidades de la aplicación. En la figura 9.1 se observa la imagen de entrada. Se ha escogido una pila debido a su sencillez para ser modelada tridimensionalmente.

Una vez cargada la imagen y establecida la posición y ángulo de la cámara (figura 9.2), se pasa a modular tridimensionalmente la pila de la imagen. Para este ejemplo se ha cargado un modelo precalculado desde disco. En las figuras 9.3, 9.4 y 9.5, se pueden observar cada una de las partes del modelo, y finalmente en la figura 9.6 se observa el modelo completo.

A continuación se procede a realizar el punteado de la imagen. En la figura 9.7 se puede observar el resultado del proceso de punteado usando la configuración por defecto y el algoritmo de Stucki como método de tramado. La configuración por defecto implica un factor de empaquetamiento igual a 3, un porcentaje de punteado de la parte modelada de la imagen de un 75 % y un 15 % de la parte no modelada; el método de detección de bordes es Sobel, y no se aplica ninguna perturbación a la posición de los puntos.

Por supuesto, esta imagen punteada se puede mejorar. Un artista notaría que los materiales de que se compone la pila son distintos. El recubrimiento



Figura 9.1: Imagen a puntear (entrada)

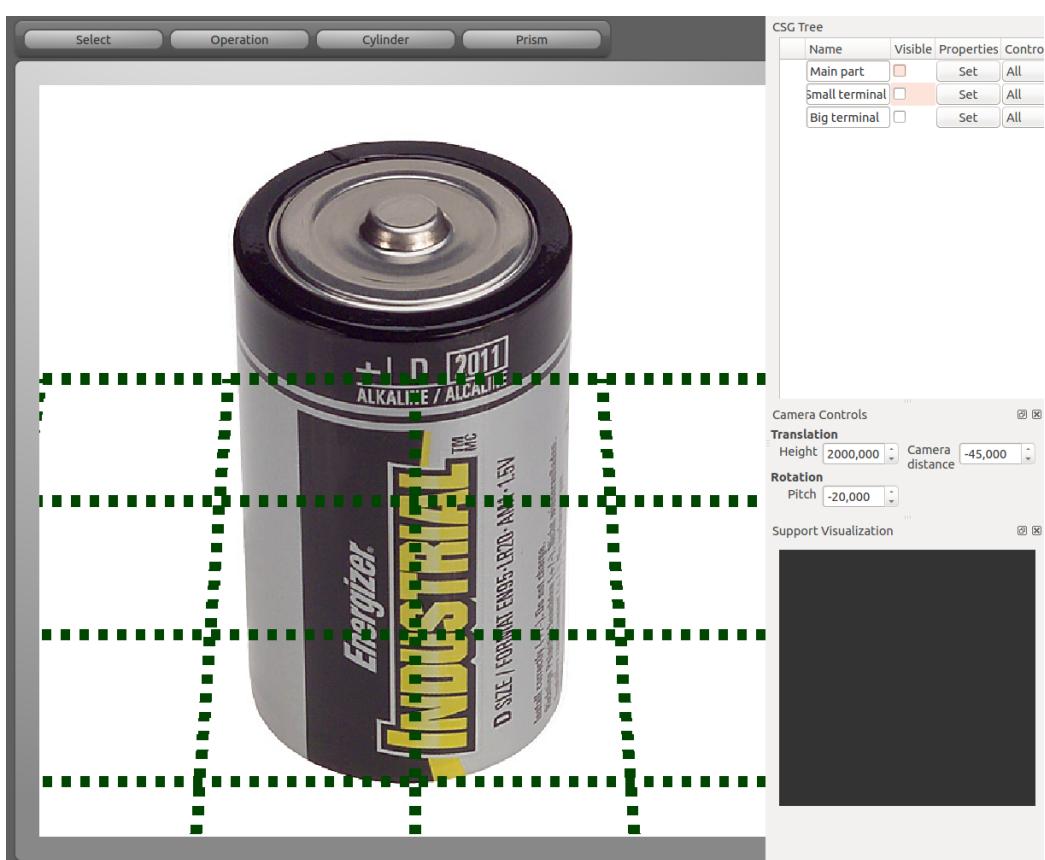


Figura 9.2: Captura de la aplicación tras establecer la cámara

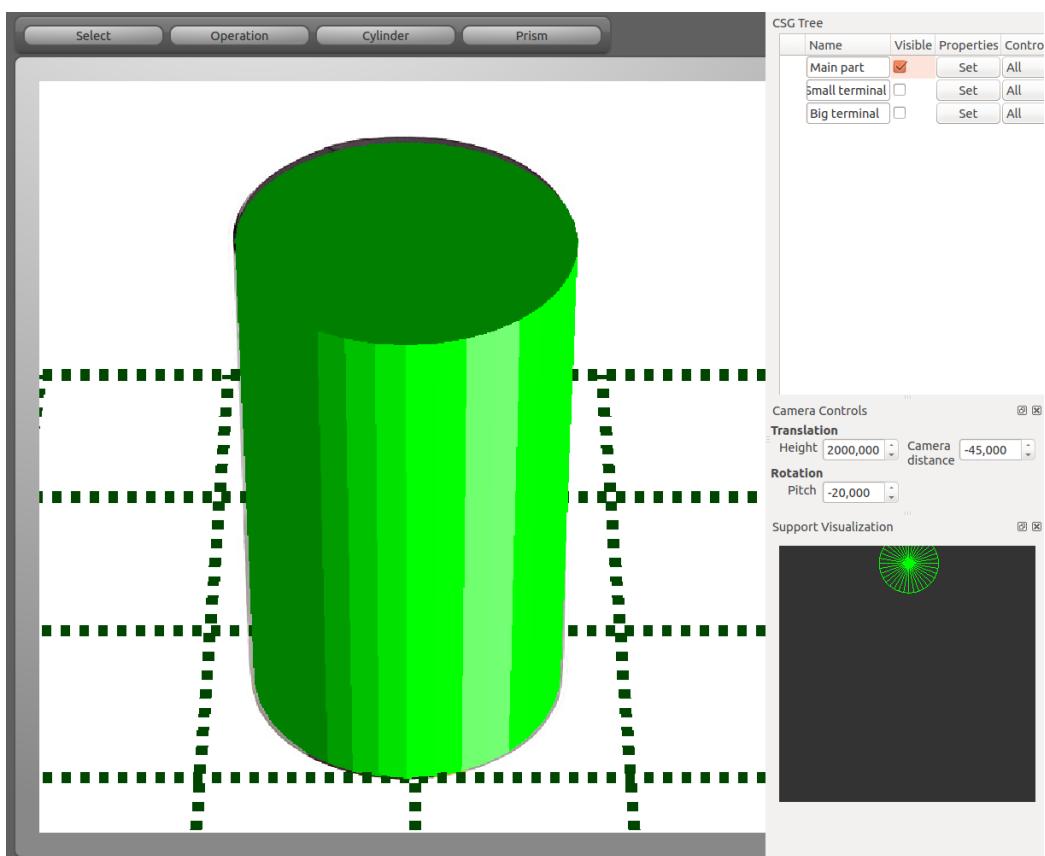


Figura 9.3: Captura de la aplicación mostrando el cilindro principal

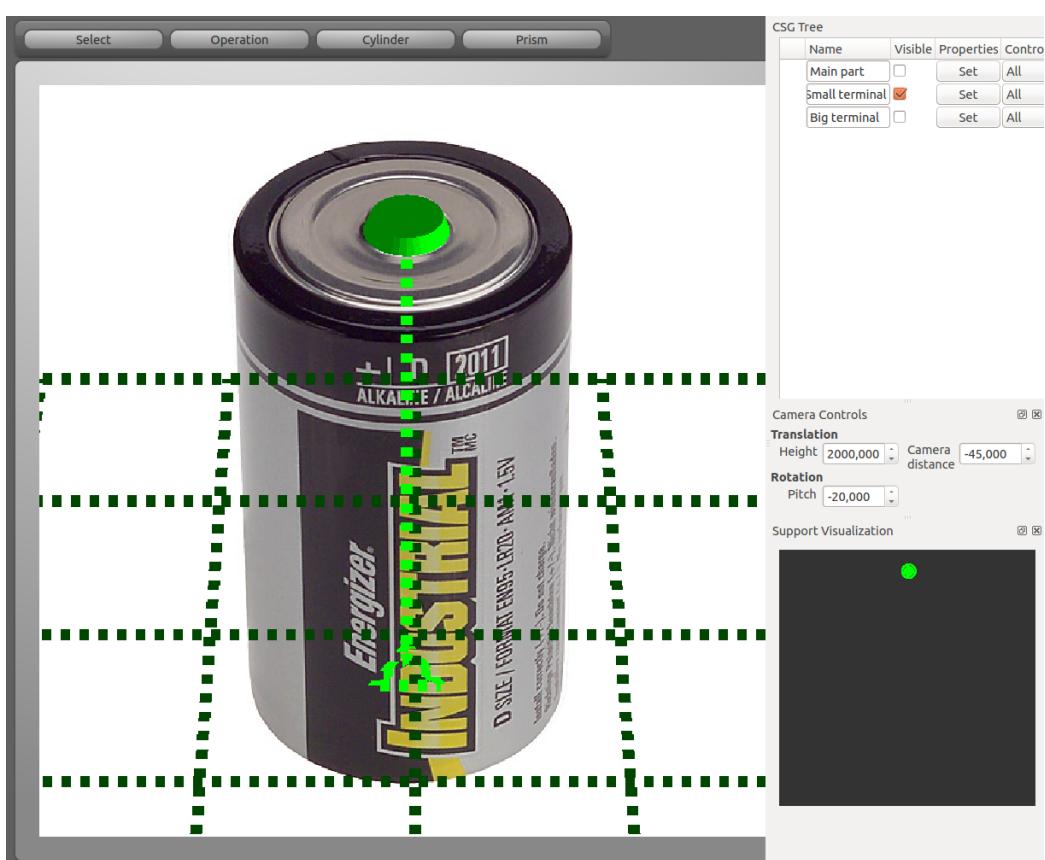


Figura 9.4: Captura de la aplicación mostrando el conector pequeño

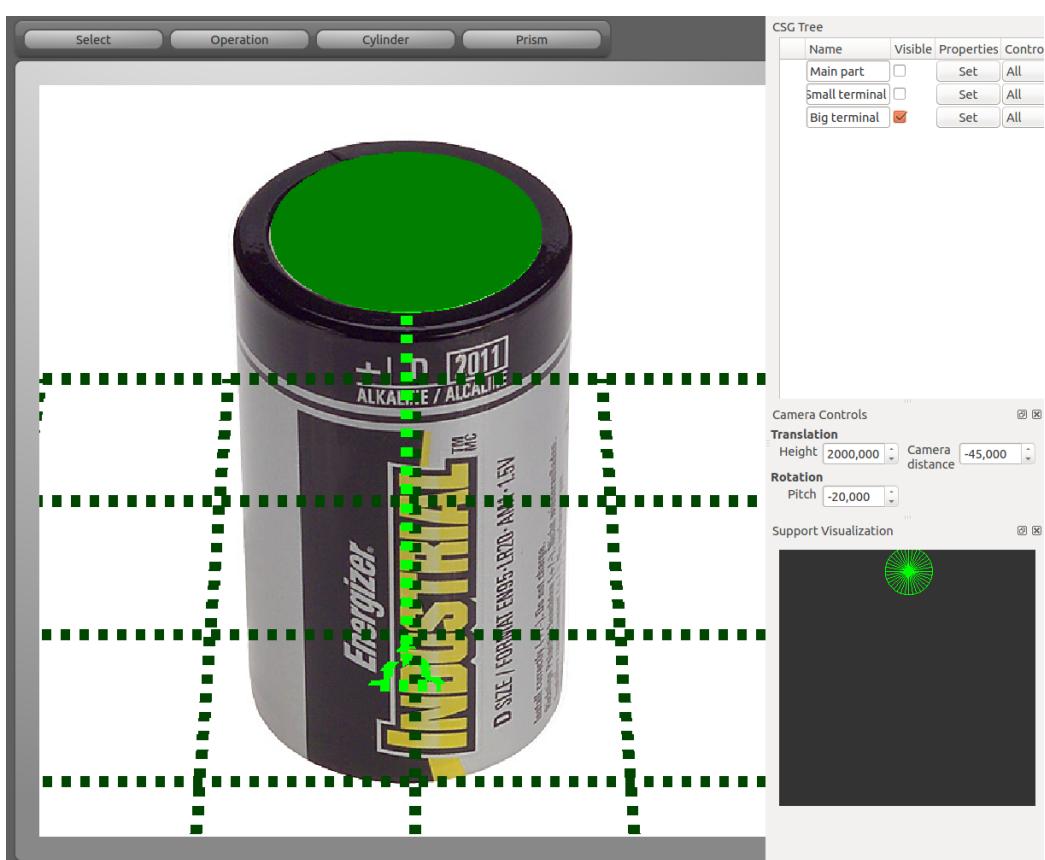


Figura 9.5: Captura de la aplicación mostrando el conector grande

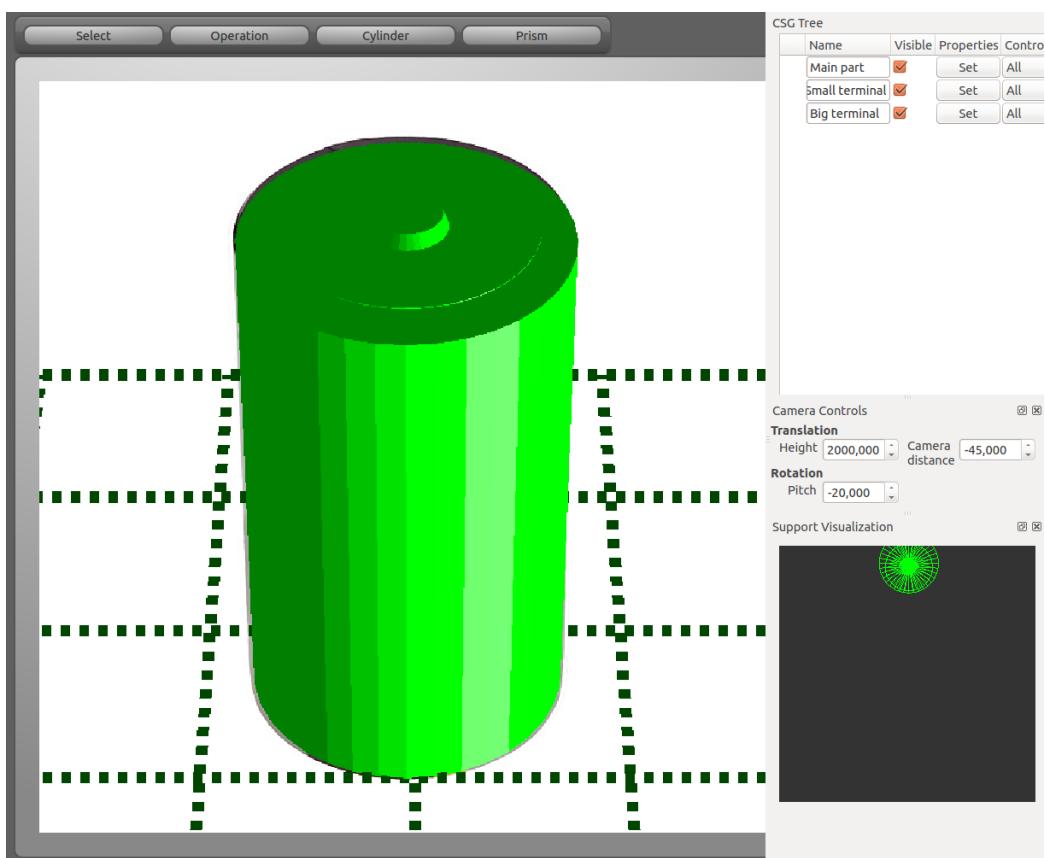


Figura 9.6: Captura de la aplicación mostrando el modelo completo

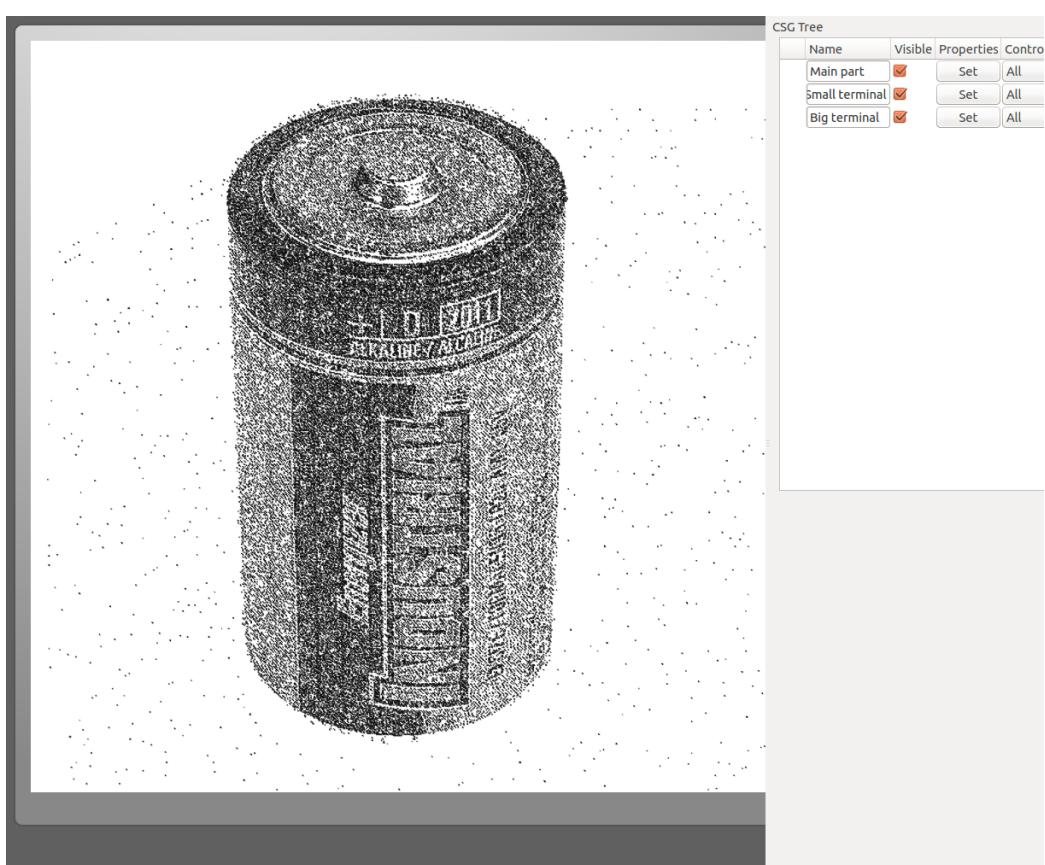


Figura 9.7: Imagen punteada usando el método de Stucki y la configuración por defecto

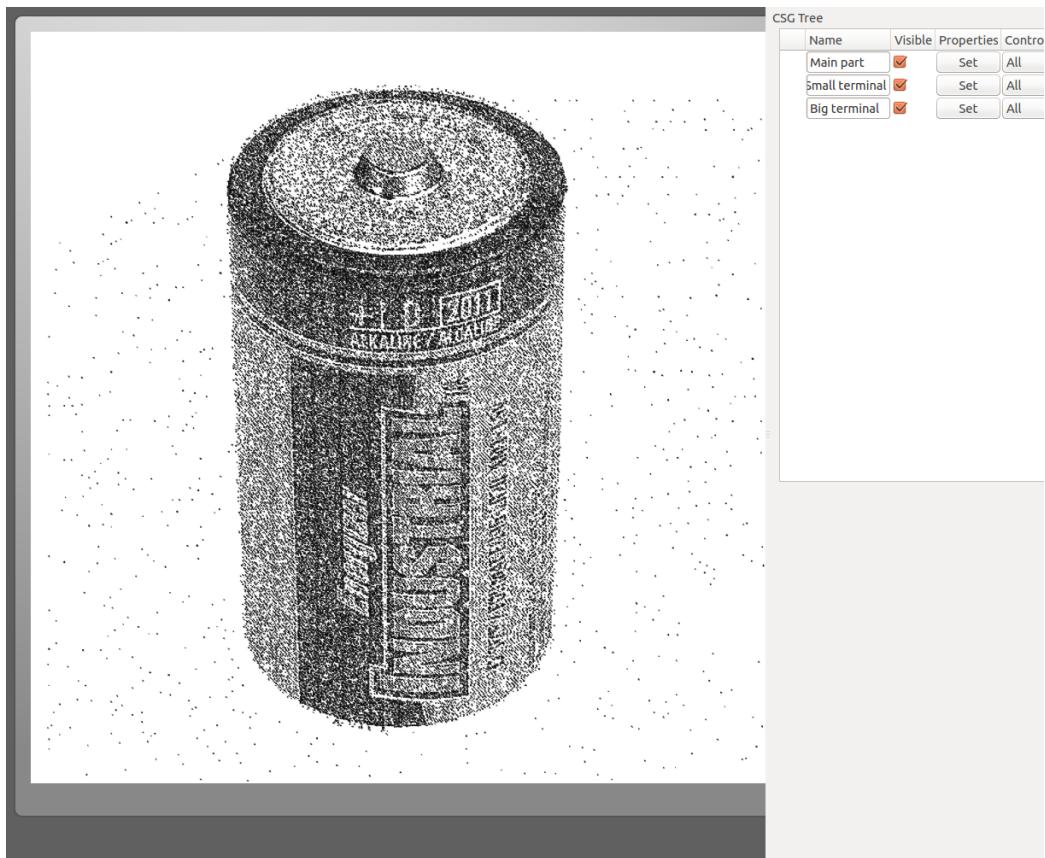


Figura 9.8: Imagen punteada generando menos puntos en la parte grande del terminal

de la mayor parte del cilindro principal es mate, pero sin embargo el terminal de la parte superior es metálico, y brilla. Para simular este efecto, un artista puntearía las zonas metálicas con menos puntos para simular el brillo.

En las figuras 9.8 y 9.9 se pueden observar ambas partes del terminal metálico superior punteadas al 50 %, de forma que, al generarse menos puntos en esas áreas, el resultado resalte más el metal brillante del que están hechas.

En la figura 9.10 se observa el resultado de reducir la generación de puntos no pertenecientes al modelo al 0 %. Así se eliminan puntos que no contribuyen ninguna información y se limpia la imagen resultante. En la figura 9.11 se añade además una dispersión de valor 20 (2 píxeles de radio).

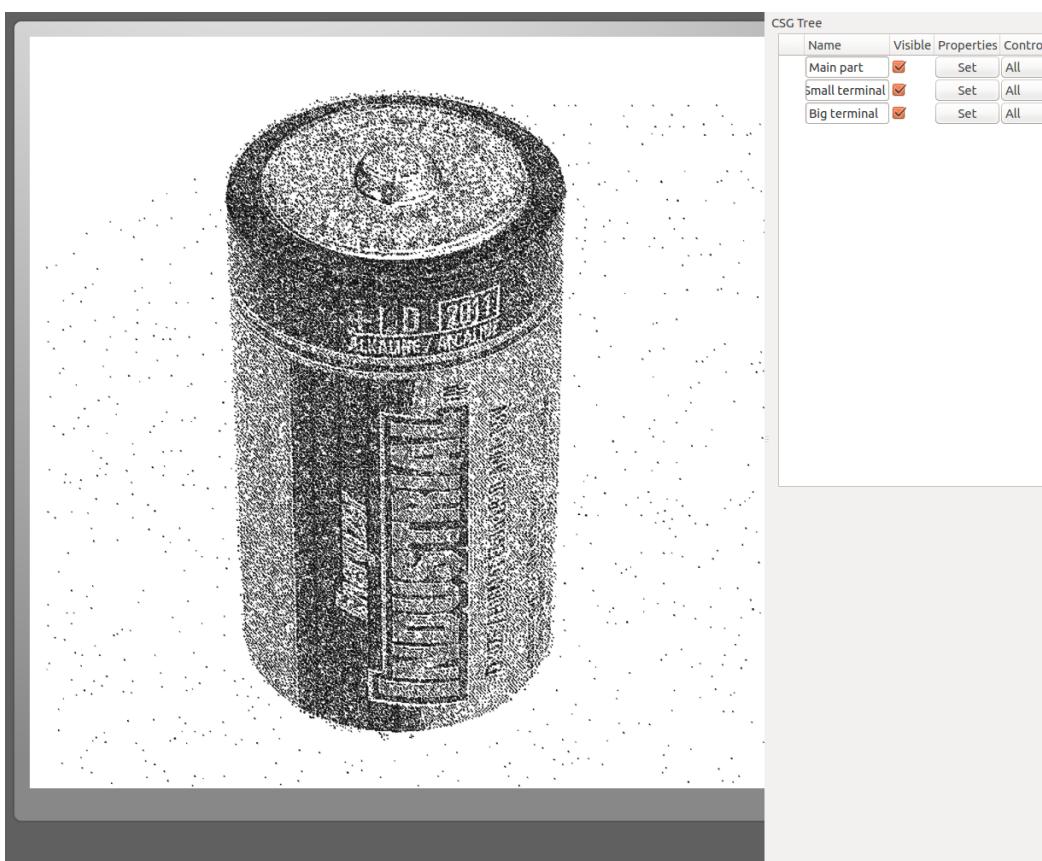


Figura 9.9: Imagen punteada generando menos puntos en ambas partes del terminal

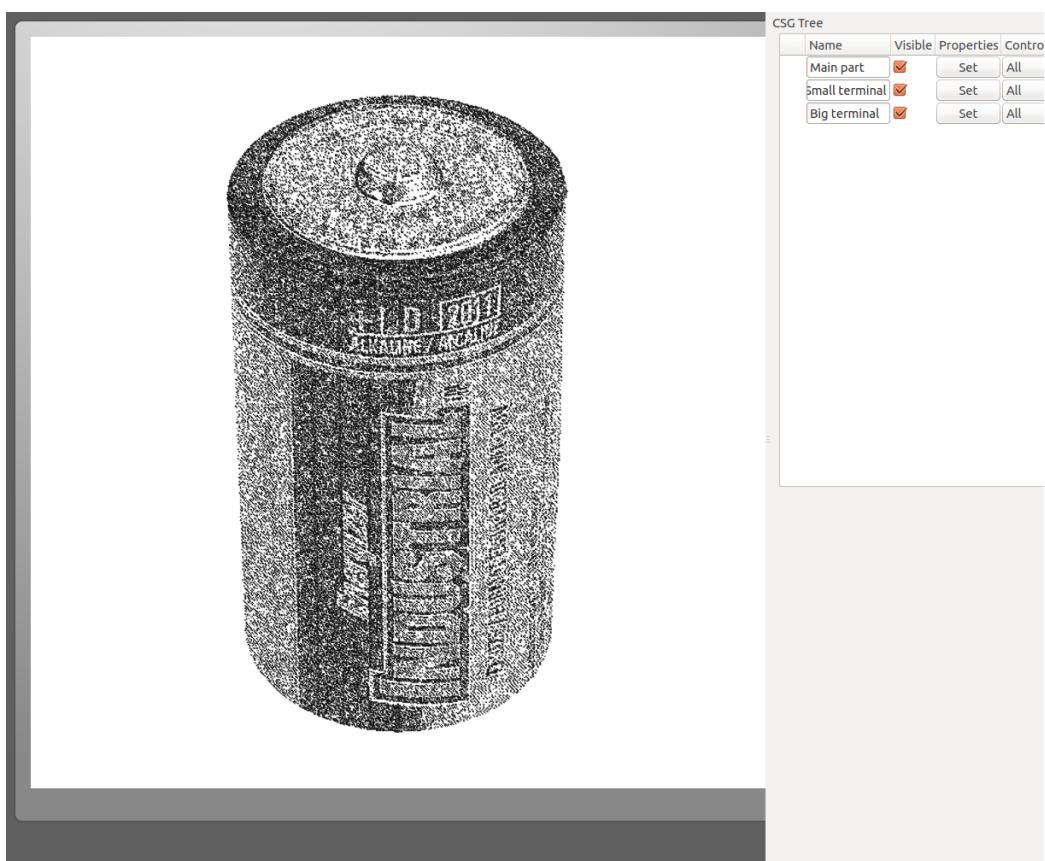


Figura 9.10: Eliminación de los puntos pertenecientes al área no modelada

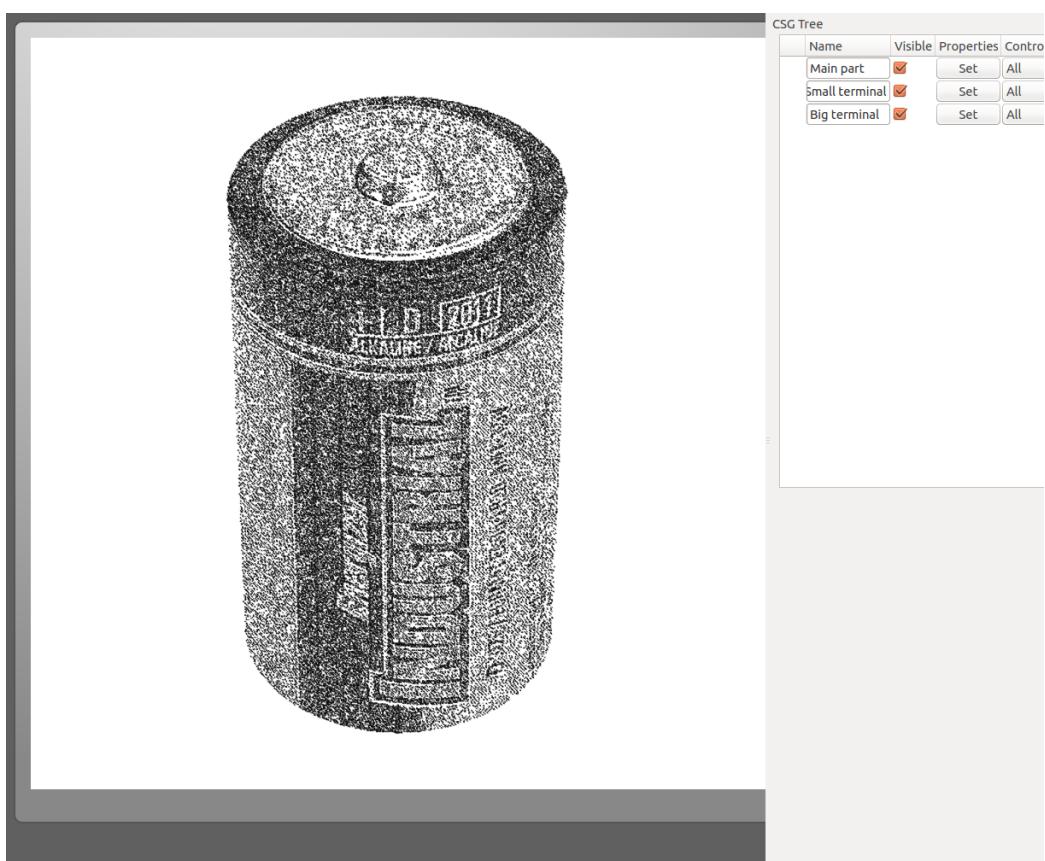


Figura 9.11: Aplicación de una perturbación a la posición de los puntos de valor 20 (2 píxeles de radio)

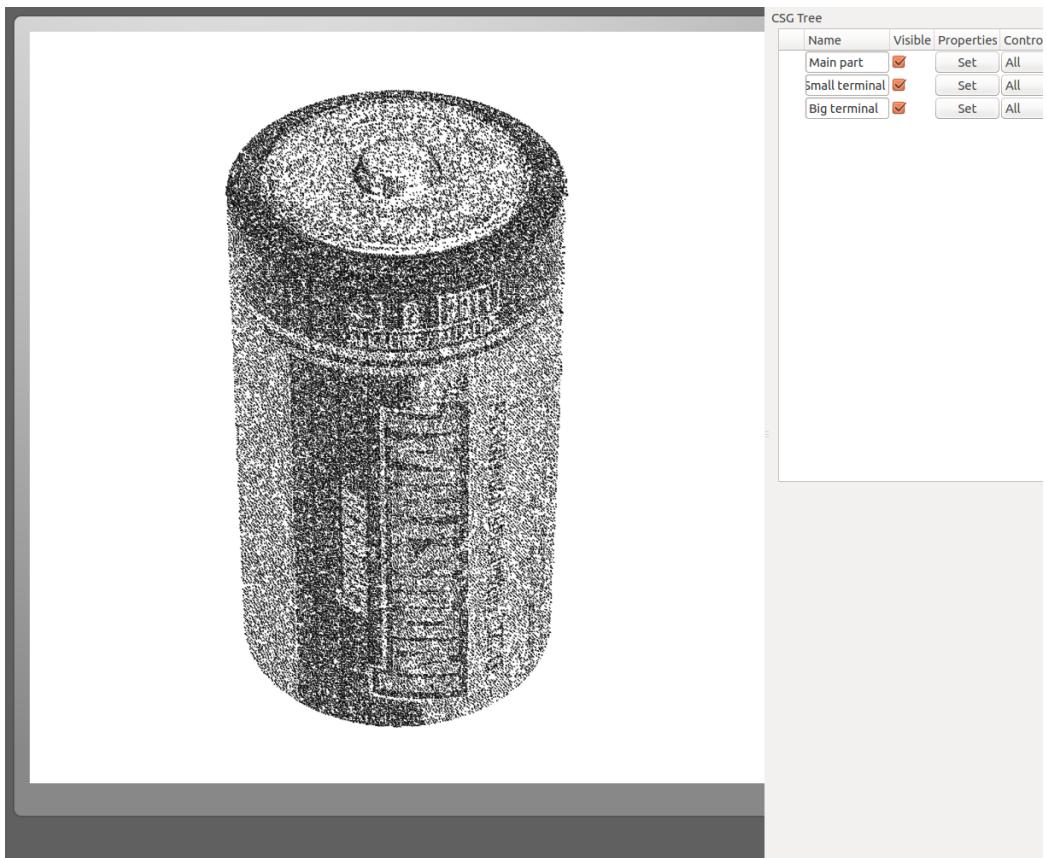


Figura 9.12: Imagen configurada de igual manera que la de la figura 9.11 pero generada usando Floyd-Steinberg en lugar de Stucki

Seguidamente, y para comparar el efecto de los métodos de tramado sobre la generación de imágenes punteadas, se añaden las figuras 9.12, 9.13 y 9.14. La figura 9.12 se ha generado usando el algoritmo de Floyd-Steinberg, y las mismas opciones de configuración que la figura 9.11. En las figuras 9.13 y 9.14 se pueden observar los resultados directos de aplicar sendos métodos de tramado a la imagen de entrada. Como ya se explicó en la sección 4, los resultados obtenidos mediante Stucki son más detallados y menos ruidosos.

A continuación se presenta una comparación entre los métodos de detección de bordes y su efecto sobre la imagen punteada. En las figuras 9.15 y 9.16 se observa el efecto que produce puntear únicamente los bordes de la parte pequeña del terminal con ambos métodos.



Figura 9.13: Imagen de entrada tramada mediante Floyd-Steinberg



Figura 9.14: Imagen de entrada tramada mediante Stucki

En la figura 9.17 se observa el una imagen en que se han punteado únicamente los bordes del modelo usando el método de Sobel. La figura 9.18 sin embargo lo hace mediante el método de Canny. Se puede observar que el método de Canny produce bordes más finos. Para la aplicación de la detección de bordes se han entregado a los métodos parámetros que produzcan resultados medios. Sería interesante permitir al usuario definir dichos parámetros para obtener una mayor o menor cantidad de bordes.

En la figura 9.19 se observa el efecto de una de las opciones de configuración, se permite la perturbación de la posición (en este caso de valor 50, o 5 píxeles de radio) de los puntos de la silueta extraída mediante detección de bordes (a menos que se especifique en la configuración de una parte del modelo, el porcentaje de puntos de la silueta que se pueden perturbar es cero).

En la figura 9.20 se puede observar el resultado de aplicar una perturbación exageradamente grande a la posición de los puntos (valor 100, o 10 píxeles de radio). Como es de esperar, se emborrona la imagen subyacente, pero la posición de los puntos ya no revela que originalmente los puntos estaban organizados en la rejilla resultante de aplicar un método de tramado.

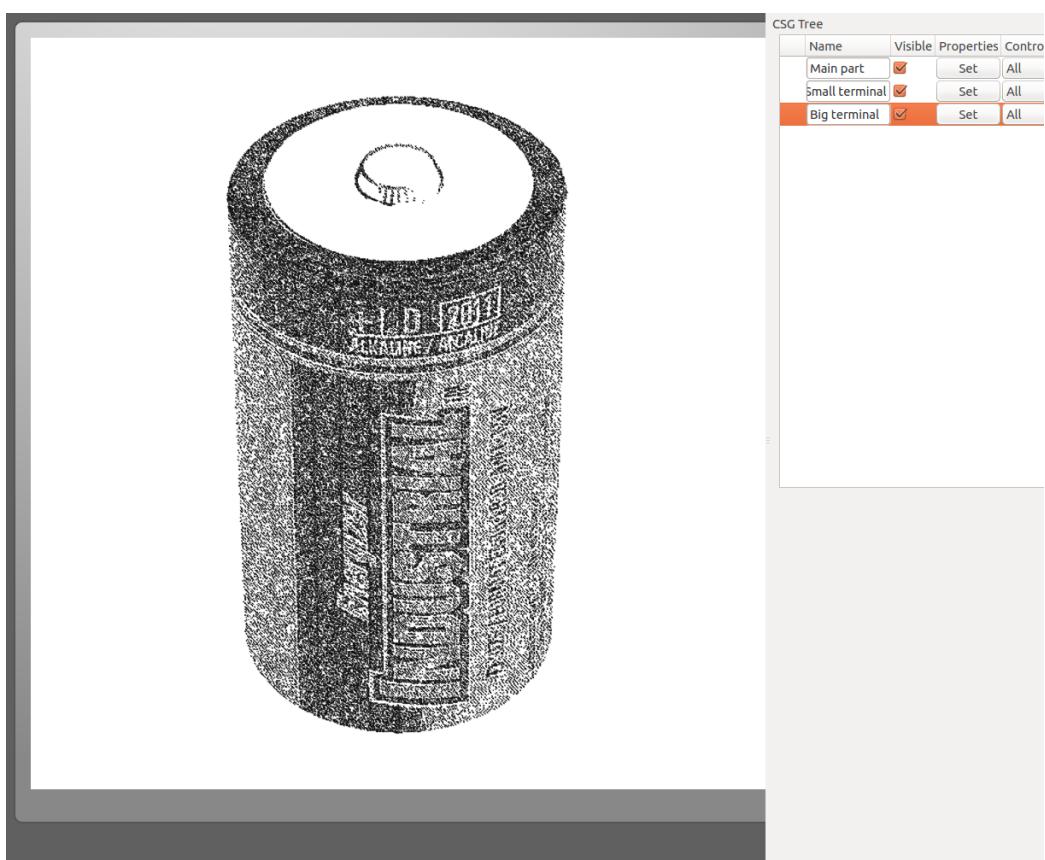


Figura 9.15: Detalle del efecto sobre la imagen punteada de los bordes generados mediante el filtro de Sobel

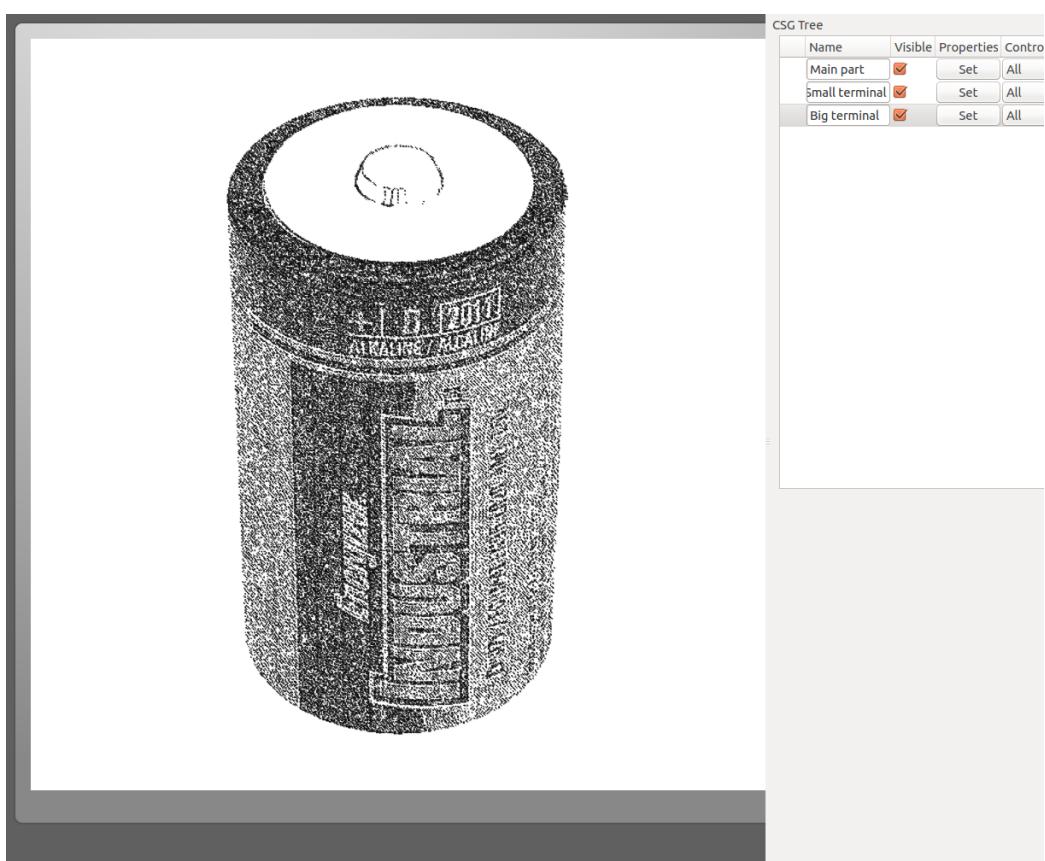


Figura 9.16: Detalle del efecto sobre la imagen punteada de los bordes generados mediante el filtro de Canny

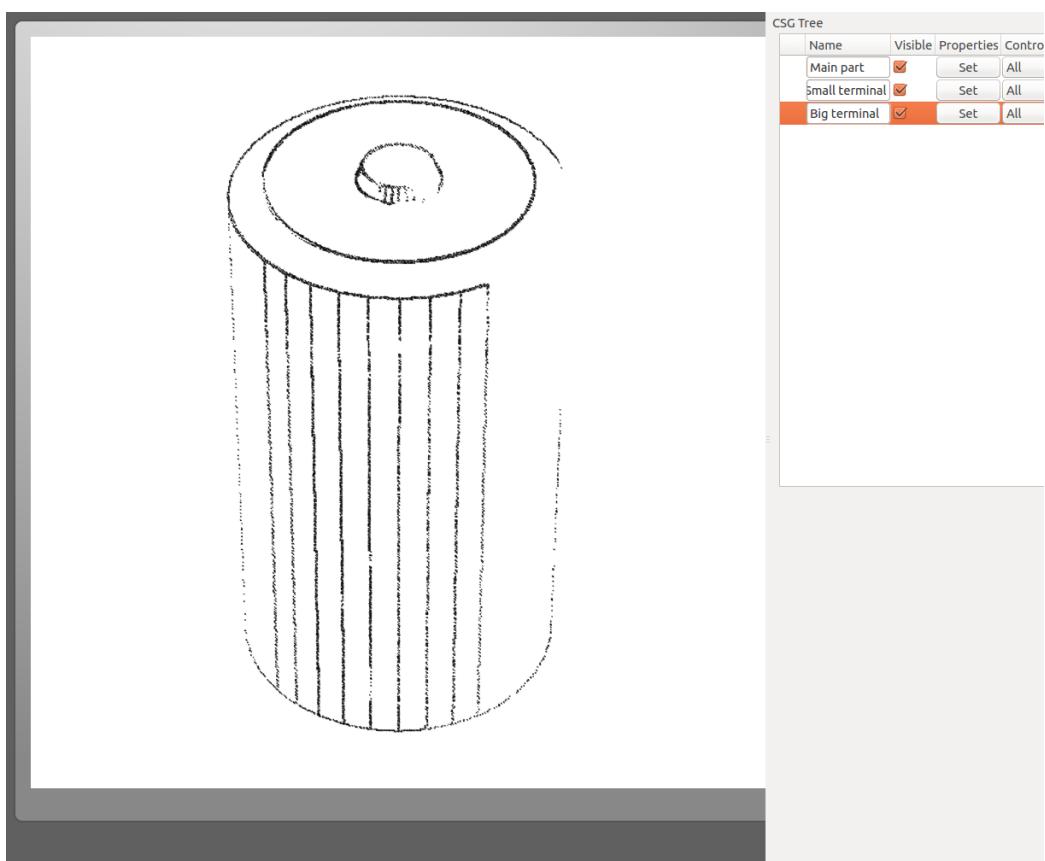


Figura 9.17: Bordes detectados en el modelo mediante el filtro de Sobel

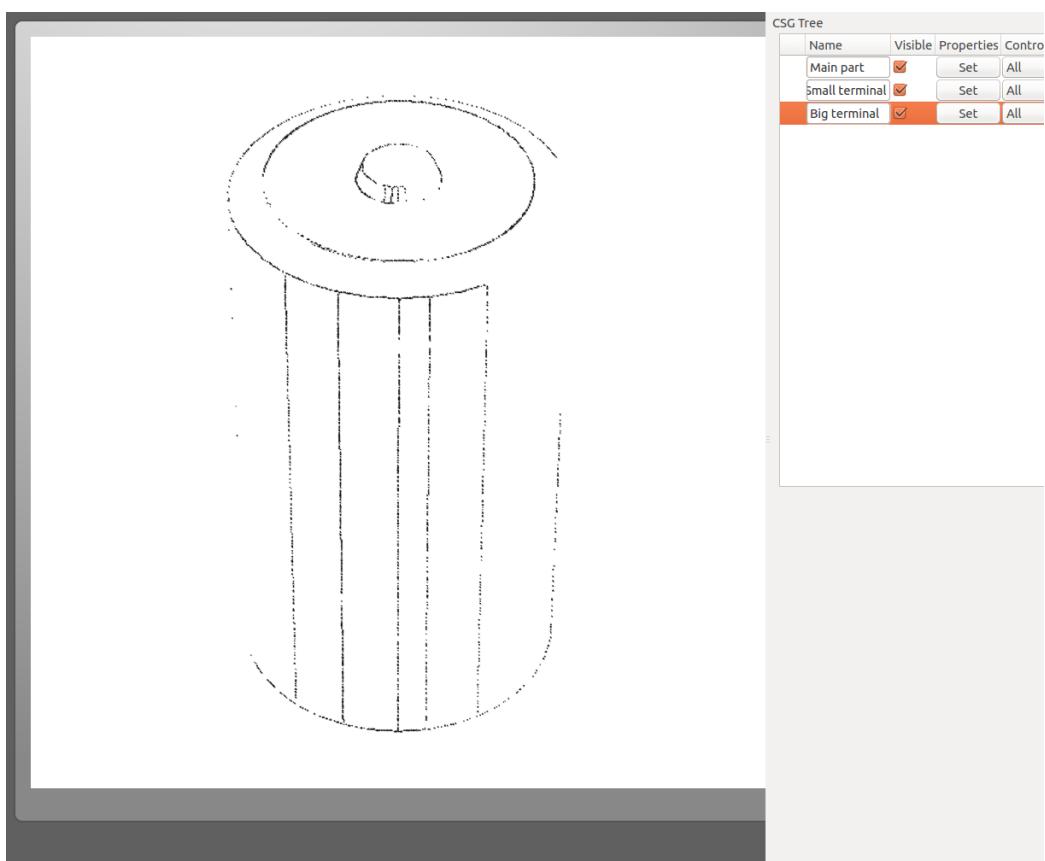


Figura 9.18: Bordes detectados en el modelo mediante el filtro de Canny

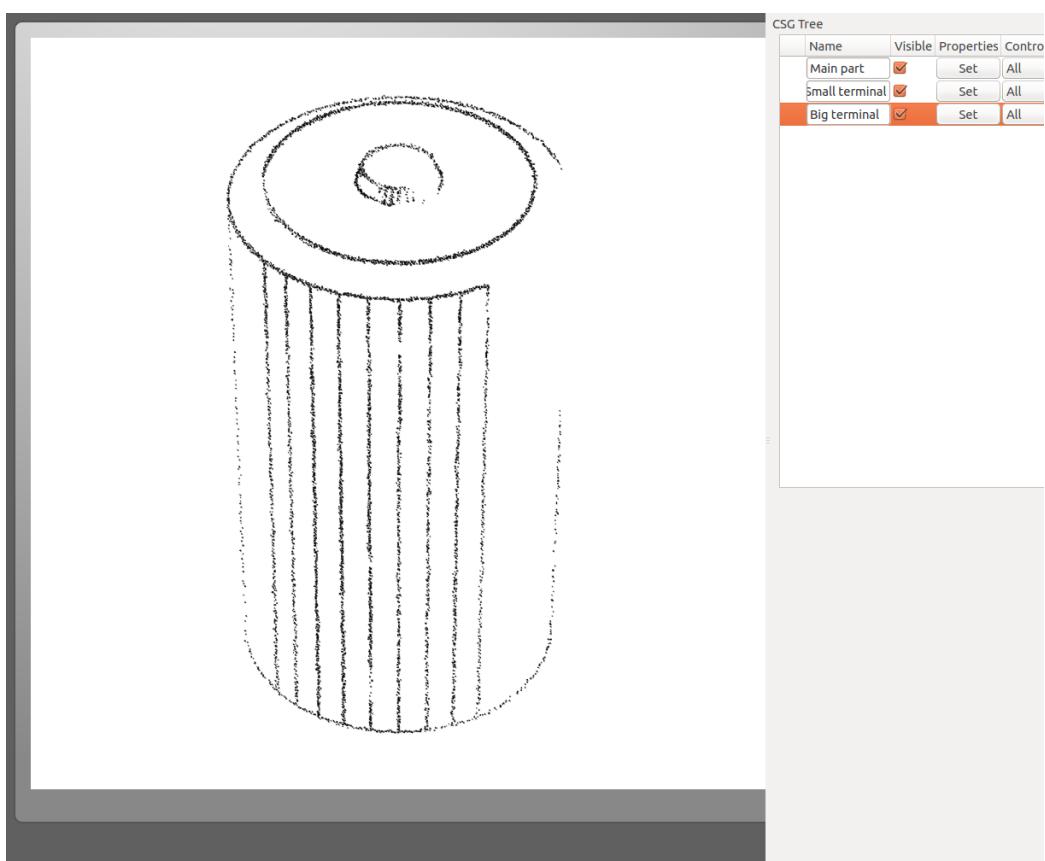


Figura 9.19: Bordes detectados en el modelo mediante el filtro de Sobel a los que se les ha aplicado una perturbación de valor 50 (5 píxeles de radio)

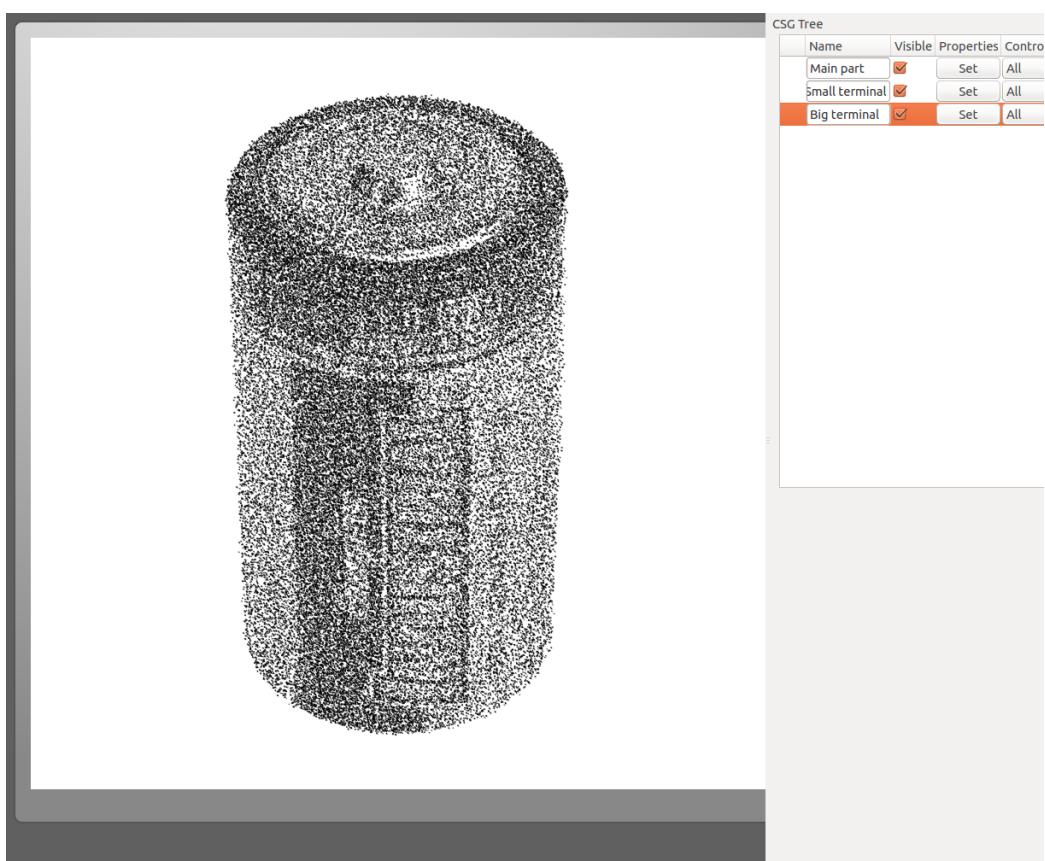


Figura 9.20: Imagen punteada mediante el método de Stucki a la que se le ha aplicado una perturbación de la posición de los puntos de valor 100 (10 píxeles de radio)

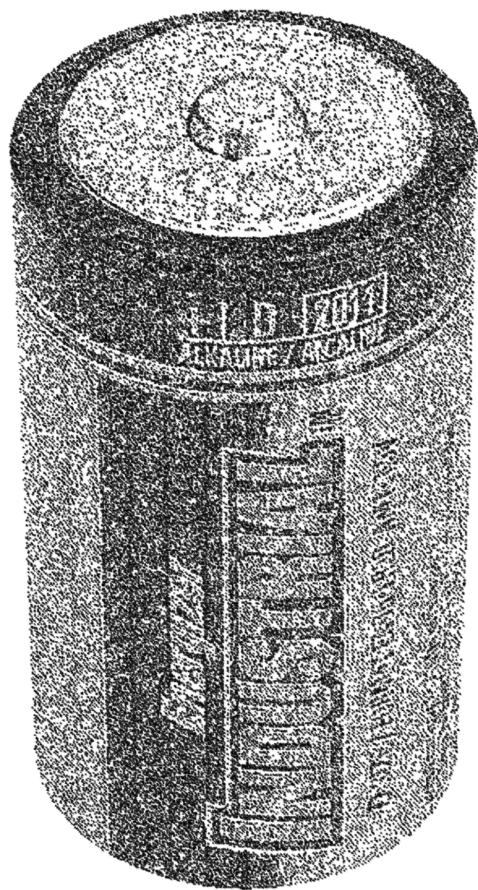


Figura 9.21: Imagen punteada generada a 1200dpi, con una resolución de 2133x2133 píxeles.

Finalmente, en la figura 9.21 se observa una imagen final generada a archivo. Para su generación se ha utilizado una configuración específica, de forma que ambas partes del terminal queden punteadas al 35% y se aplica una dispersión de valor 25 (2.5 píxeles de radio). En esta versión se ha punteado el terminal con menos puntos para aumentar el brillo y dar una apariencia más metálica.

Capítulo 10

Manual de usuario

La aplicación tiene esencialmente dos modos: el motor 3D que permite crear modelos tridimensionales usando operaciones booleanas, y el motor de stippling, que transforma la imagen original en una imagen punteada, y permite aprovechar la información tridimensional proporcionada en el motor 3D.

Para ello, el primer paso consistirá en cargar una imagen mediante el menú *File/Open Image*, que abrirá una vista del disco duro y permitirá escoger la imagen de entrada. La aplicación intentará entonces leer los metadatos de la imagen, y reconstruir automáticamente la escena haciendo uso de las características de la cámara y la distancia focal en el momento de tomar la fotografía. En caso de que no se encuentre la distancia focal, aparecerá una ventana que nos permitirá definir los principales parámetros de la escena. Una vez cargada la imagen, se puede cambiar la cámara, y con ello la escena, o incluso definir una cámara especificando sus parámetros mediante el menú *Edit/Camera Model*, que contiene una lista de las cámaras conocidas por el programa.

Una vez cargada la imagen con éxito, aparecerá en una ventana interna, donde aparece representado el suelo y según se interaccione, las entidades tridimensionales creadas. En el lateral derecho, aparecerá una barra con el árbol de sólidos primitivos y operaciones booleanas (árbol CSG), un menú con los parámetros de control de la cámara (ángulo de inclinación, altura y distancia) y una visualización de apoyo que representa la escena desde un punto de vista superior.

Esta vista interna soporta operaciones típicas de zoom (mediante la rueda

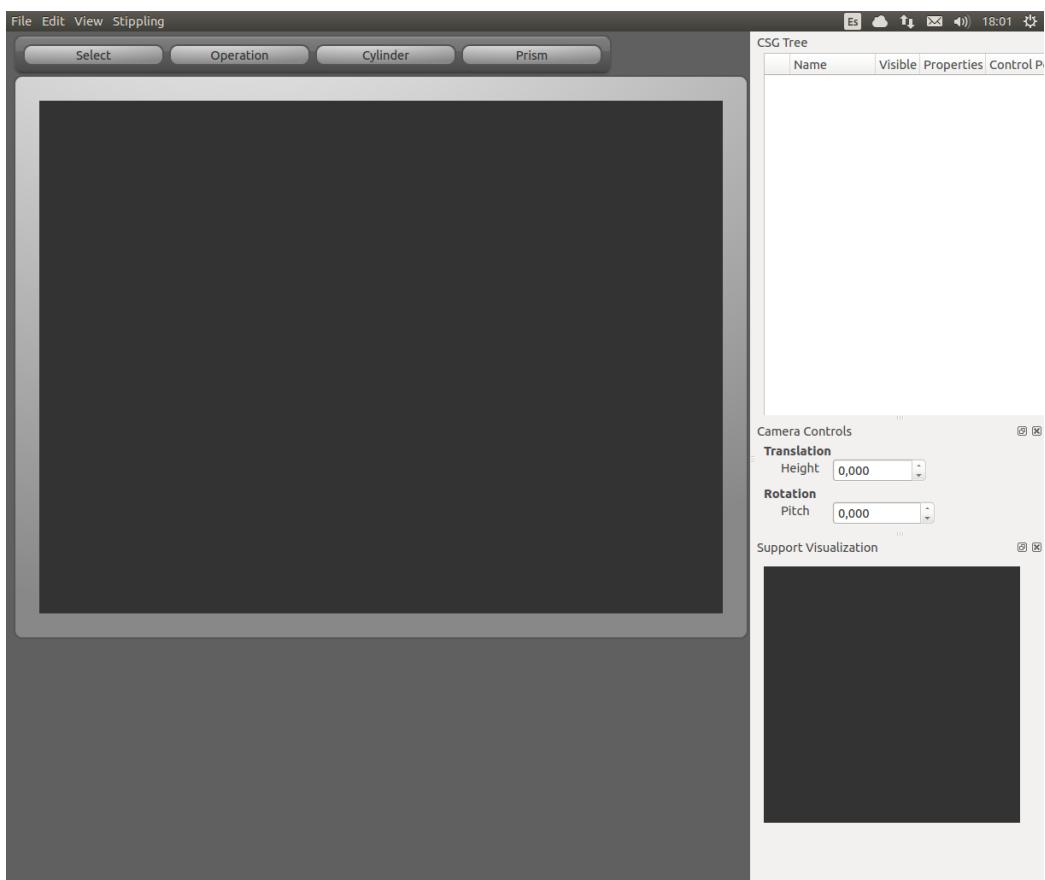


Figura 10.1: Ventana principal en el modo de motor tridimensional (sin imagen cargada).

CSG Tree					
	Name	Visible	Properties	Control Points	Type
▼	Operation 34	<input checked="" type="checkbox"/>	Set	All	Op. Union
▼	Operation 23	<input checked="" type="checkbox"/>	Set	All	Op. Union
	Prism 19	<input checked="" type="checkbox"/>	Set	All	Primitive
	Cylinder 12	<input checked="" type="checkbox"/>	Set	All	Primitive
	Cylinder 30	<input checked="" type="checkbox"/>	Set	All	Primitive

Figura 10.2: Detalle del widget que contiene la vista del árbol CSG.

del ratón) y desplazamientos (arrastrando y soltando con el botón derecho del ratón), y en caso de desechar volver a la visualización por defecto (imagen centrada y totalmente visible), no hay más que recurrir al menú *View/Reset viewing transformations*.

A través del menú *File*, se podrá importar y exportar el árbol CSG (*File/Import CSG Tree* y *Export CSG Tree*), facilitando el desarrollo de objetos complejos. Para añadir entidades, se deberá usar la barra que se encuentra bajo el menú y sobre la ventana interna mencionada anteriormente. En dicha barra aparecen las opciones para seleccionar o añadir entidades (primitivas u operaciones).

Para añadir entidades, basta con seleccionar la opción deseada y mover el puntero a la ventana interna, donde aparecerá la entidad colocada sobre el suelo. Al hacer clic, la entidad fija su posición y se podrá empezar a interactuar con ella. Las entidades se pueden eliminar seleccionándolas y pulsando la tecla *Supr*. Las interacciones se realizan mediante puntos de control, y para ello es necesario seleccionar la entidad (haciendo clic sobre ella, o seleccionándola en la vista lateral de árbol). Los puntos de control representan diversas interacciones:

- Variación de posición (punto de control rojo por defecto).
- Rotación sobre el eje Y (punto de control negro por defecto).
- Variaciones de la geometría de la entidad (puntos de control azules por defecto).

Las interacciones son muy directas e intuitivas, y sólo moviendo el ratón mientras se arrastra el punto de control se observan directamente los resultados. La interacción de variación de posición tiene una pequeña complicación extra, en el caso de querer mover la entidad sobre el plano del suelo, no es necesario hacer nada más que interactuar con el punto de control con normalidad, pero en el caso de querer cambiar el plano sobre el que se desplaza (más alto o más bajo), se deberá mantener pulsada la tecla *Ctrl* mientras se mueve el ratón a la altura deseada.

En la vista lateral de árbol CSG, se pueden seleccionar entidades, darles nombre, establecer propiedades de punteado específicas, establecer su visibilidad, escoger los puntos de control visibles y, en el caso de las operaciones, escoger el tipo de operación booleana.

Las operaciones reciben sus operandos mediante el uso de esta vista. Simplemente se debe coger el elemento de la vista cliqueando sobre él y, manteniendo pulsado, arrastrarlo sobre la operación deseada, momento en que se dejará de cliquear. El orden de los operandos, según la operación, puede alterar el

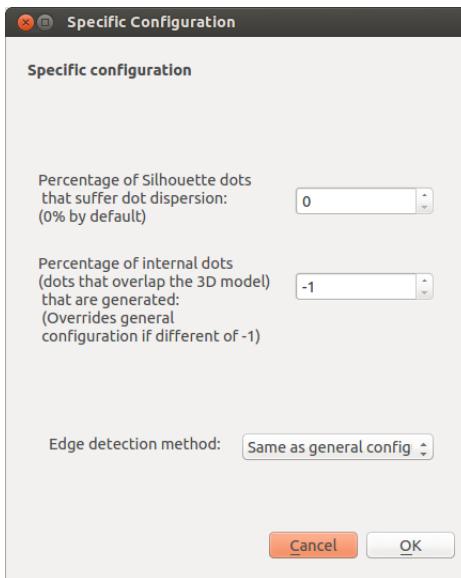


Figura 10.3: Detalle del diálogo de configuración específica de un modelo.

resultado, y para cambiar el orden, no hay más que sacar el operando deseado de la operación usando el mismo método, para luego añadirlo de nuevo.

En el menú *View*, se pueden encontrar múltiples opciones de visualización, tanto de las entidades como del suelo, pudiendo representarse como rejillas, sólidos o sólidos iluminados en el caso de las entidades, o como rejilla, sólido, sólido y rejilla u oculto en el caso del suelo.

Mediante el menú *Edit/Configuration* se puede acceder a la configuración general del sistema, pudiendo cambiar diversas opciones de configuración, tanto de renderización, como de punteado.

En esta pantalla se pueden cambiar los colores que se usan en la ventana interna, de manera que si una fotografía produce un bajo contraste con la configuración por defecto, se pueda cambiar para poder trabajar cómodamente.

Una vez creado un modelo, antes de pasar al modo de stippling, se debe revisar la configuración general, en el apartado de stippling, para asegurar el resultado deseado sin necesidad de esperar para recalcular la imagen punteada. También se deben revisar las opciones de configuración específicas de las entidades, pudiendo puntear unas zonas de forma distinta a otras.

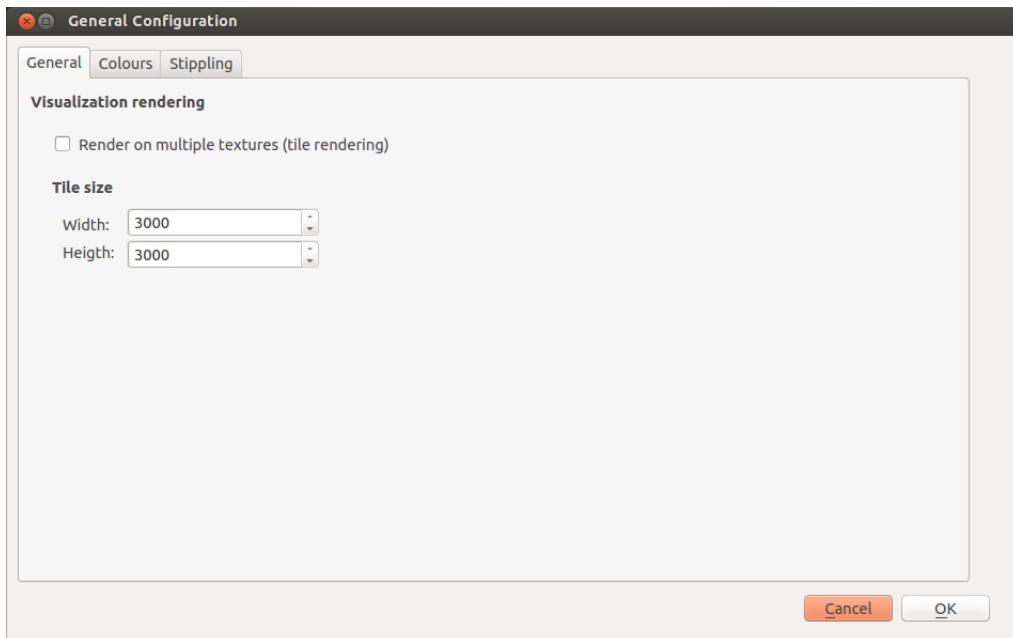


Figura 10.4: Detalle del diálogo de configuración general (pestaña general).

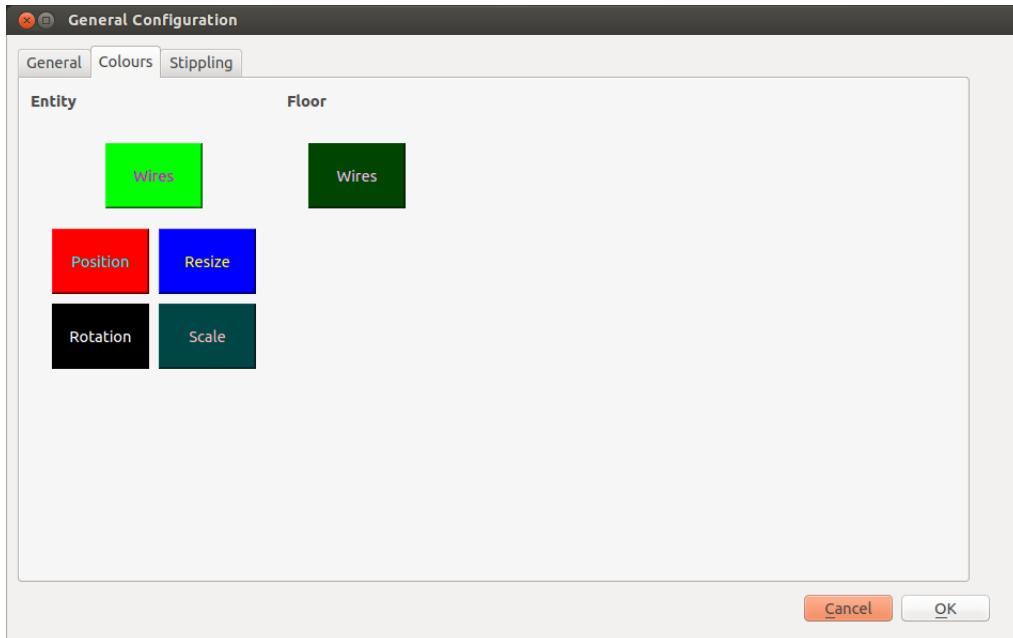


Figura 10.5: Detalle del diálogo de configuración general (pestaña colores).

Una vez se esté satisfecho con la configuración, mediante el menú *Stippling/-Generate and render* y escogiendo el método de trámado entre los métodos de Floyd-Steinberg o de Stucki (el de mayor calidad es el Stucki, pero es

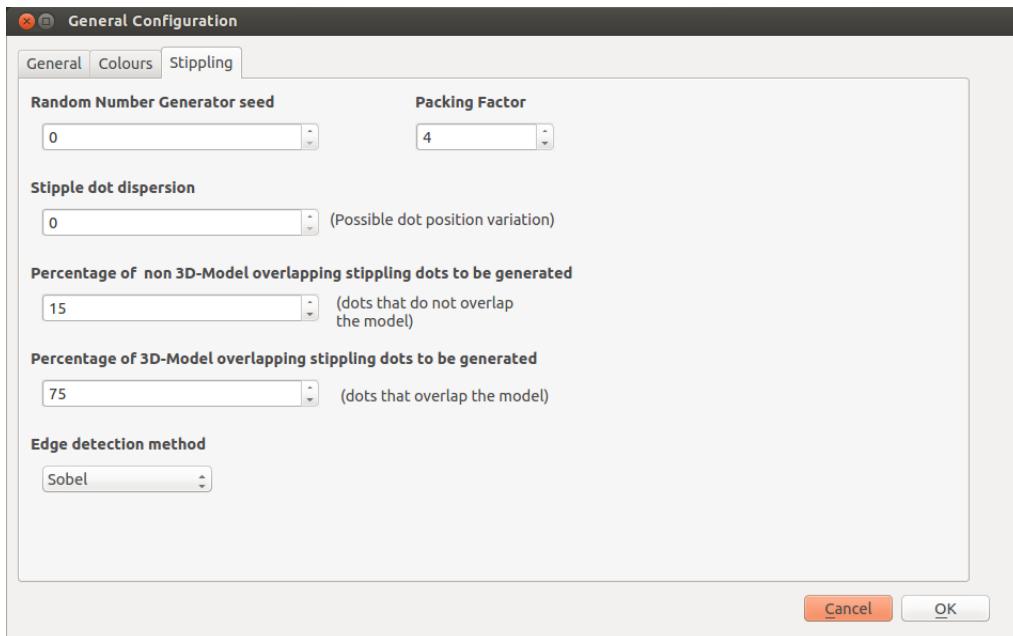


Figura 10.6: Detalle del diálogo de configuración general (pestana stippling).

también más lento). Tras escoger el método se genera la imagen punteada (el proceso es largo y complejo, en particular si la imagen original es grande, así que se debe tener paciencia). Una vez terminado el proceso, la imagen punteada aparecerá sobre la ventana interna, y se podrá hacer zoom (rueda del ratón) y desplazar la imagen (manteniendo pulsado el botón derecho mientras se mueve el ratón en la dirección deseada).

En caso de que se desee aplicar una perturbación (por defecto es cero, pero se puede especificar en la configuración general) a la posición de los puntos, se puede hacer mediante el menú *Stippling/Re-apply Stipple Dot dispersion*. Si la perturbación es suficientemente grande, la imagen representada empezará a emborronarse, así que debe aplicarse con mesura.

Es importante notar, que para que los cambios de configuración tengan efecto (si ya se ha generado una imagen punteada), se deberá pedir al programa que regenere la imagen punteada, o que re-aplique la perturbación a los puntos, según sea el caso. Por ejemplo, tras cambiar el factor de empaquetamiento en la configuración general, la imagen punteada no cambiará automáticamente, sino que será necesario regenerarla mediante el menú *Stippling/Generate and render* y escogiendo el método de tramado, tal como se hizo al generar la imagen punteada por primera vez.

Finalmente, para generar la imagen final (pasar la imagen a disco) se debe usar el menú *Stippling/Generate final image*, que hará aparecer un diálogo para escoger el lugar y nombre con que se guardará la imagen png a generar. De nuevo, este es un proceso largo y costoso, especialmente dependiente del tamaño de la imagen original, así que se recomienda una buena dosis de paciencia, aún cuando la aplicación parezca bloqueada.

Capítulo 11

Manual de compilación

El proyecto es multiplataforma, y se incluye un manual de compilación probado en sistemas Linux y Windows.

Para ello se ha utilizado la herramienta de generación de código de compilación CMake, por su capacidad de generar código compatible para cualquier sistema (aunque aquí no se detalle como, también se puede compilar bajo Mac, y de forma muy similar).

El primer paso para poder compilar, consiste en instalar las siguientes bibliotecas:

- Qt 4.8
- OpenGL 3.3 o superior
- GLEW
- GLM
- OpenCV 2.0 o superior
- CGAL
- Exiv2

Dichas bibliotecas se encuentran en el dvd en el directorio /dependencies/. En caso de usar linux, se recomienda encarecidamente usar el gestor de paquetes para instalar las bibliotecas debido a la simplicidad de uso del mismo.

Es necesario reiterar la importancia de disponer de una tarjeta gráfica que

soporte OpenGL 3.3 o superior. Por supuesto, además se necesitará un compilador de C++.

Una vez que dichas bibliotecas hayan sido instaladas (siguiendo sus correspondientes tutoriales específicos para el sistema operativo elegido), dependiendo del sistema operativo en que se hayan instalado, puede que necesiten algunas configuraciones adicionales.

Como es natural, habrá que copiar la carpeta que contiene el código del proyecto (/source/stippling/) a disco, pues el DVD no es modifiable.

- Linux

Suponiendo que las bibliotecas hayan sido instaladas y enlazadas dinámicamente (forma de instalación típica cuando se descargan paquetes mediante el gestor de paquetes) entonces todo está listo para funcionar. Sin embargo, si las bibliotecas han sido compiladas a mano, e instaladas a partir del código compilado, será necesario generar los enlaces de forma que el sistema operativo pueda conocer su localización en tiempo de ejecución (la documentación de compilación de cada biblioteca incluye instrucciones para este caso).

- Windows

Una vez que las bibliotecas hayan sido instaladas (es necesario prestar especial atención a la descarga de los ficheros de la biblioteca, pues deben escogerse aquellas compiladas con el compilador que se usará más adelante y la estructura correspondiente a la máquina utilizada). Además será necesario establecer algunas variables de entorno. Rapid Environment Editor (<http://www.rapidee.com/es/about>) es una herramienta gratuita que simplificará mucho el proceso.

El primer paso consiste en añadir a la variable Path los directorios que contienen los ficheros de ejecución (binaries) de cada biblioteca (este paso es parte de las instrucciones de cada biblioteca, pero se ha incluido para asegurar que se realiza correctamente).

El siguiente paso consiste en añadir las siguientes variables de entorno:

GLEW_INCLUDE_PATH, incluyendo la ruta de la instalación, por ejemplo: C:\libs\glew-1.10.0\include

GLEW_LIBRARY, incluyendo la ruta del directorio que contiene los ficheros de ejecución (binaries), por ejemplo: C:\libs\glew-1.10.0\bin\Release\x64. Es importante notar que en las últimas versiones, los ficheros dll y lib se encuentran en carpetas diferentes dentro del directorio bin, y por lo tanto, debido a la forma en que Windows trabaja, deberán ser colocados en una misma carpeta.

GLEW_ROOT_DIR, incluyendo la ruta del directorio raíz de la biblioteca GLEW, por ejemplo: C:\libs\glew-1.10.0

GLM_ROOT_DIR, incluyendo la ruta del directorio raíz de la biblioteca GLM, por ejemplo: C:\libs\glm

Finalmente, la ruta del ejecutable exiv2 deberá ser añadida a la variable Path, de forma que se pueda invocar el ejecutable exiv2.exe directamente desde consola.

Finalmente, una vez que todo se ha instalado y configurado, hay que navegar mediante la consola al directorio raíz del proyecto (el que contiene *CMakeLists.txt*) y ejecutar:

```
cmake .
```

Ese comando configura el proyecto y prepara los ficheros de compilación. Ahora, dependiendo del sistema operativo, se puede usar el comando:

```
make .
```

directamente (en *Linux*), o se creará un proyecto de Visual Studio (en *Windows*) que permitirá construir la solución (se podría usar nmake en Windows, pero eso requeriría compilar las bibliotecas a mano, complicando el proceso innecesariamente).

Y eso es todo. Una vez compilado, se habrá generado el fichero ejecutable, y se podrá ejecutar el proyecto.

Capítulo 12

Problemas y soluciones

12.1. Degeneración de poliedros de CGAL

Este problema acaeció durante la integración de CGAL para el soporte de operaciones booleanas entre sólidos. Debido a que CGAL no soporta sólidos de volumen cero (aparece una excepción por degeneración del polihedro), fue necesario introducir medidas de protección para imposibilitar la aparición de dicho caso. Para ello, se establecieron mínimos en las distancias que los puntos de control permiten definir, y ángulos mínimos y máximos para las caras del prisma. En caso de expansión del número y tipo de sólidos primitivos, será necesario considerar con especial atención este punto.

12.2. Integración de VBO/VAO y el cauce antiguo de OpenGL

Siguiendo la recomendación de Germán de que usar el cauce antiguo de OpenGL facilitaría la implementación del sistema de iluminación de sólidos, aparece el problema de que el cauce antiguo no soporta el sistema de VBO/-VAO (Vertex Buffer Object / Vertex Array Object) que estaba siendo usado hasta el momento. Tras comprobar que de hecho ya no eran compatibles, se procedió a mantener una copia en memoria RAM de los datos (mediante VBO/VAO dichos datos residirían en la memoria de la GPU) para proceder a enviarlos en cada evento de refresco a la GPU (Esta solución es más ineficiente que el sistema planteado inicialmente, pero no hubo manera de hacer funcionar el cauce antiguo con los VBO/VAO).

12.3. Mejora de la eficiencia de representación de la imagen de punteado

Debido a la gran cantidad de puntos que una imagen punteada posee, se hizo necesario optimizar la eficiencia de representación de la imagen punteada. Para ello se procedió a la implementación de un QuadTree para poder limitar el área de renderización al área necesaria en cada momento, de manera que no se representen los puntos que quedan fuera del área de interés.

12.4. Uso de renderización en textura para evitar realizar renderizaciones redundantes

A pesar de haber usado un QuadTree para reducir al mínimo el número de puntos según el área de renderizado, la eficiencia estaba aún lejos de ser aceptable, ya que en cada frame se renderizaban todos los puntos, independientemente de que el área de la imagen, o las posiciones de los puntos hubiesen cambiado. Para solucionar este problema, se procedió a realizar una renderización sobre textura, de manera que sólo se realizase la renderización de los puntos cada vez que ocurriese algún cambio, y entre cambios, sólo fuese necesario renderizar los resultados calculados anteriormente y almacenados en una textura.

12.5. Uso de múltiples texturas para evitar aberraciones producidas por la interpolación realizada sobre texturas.

Al renderizar sobre una única textura, aparecen diferencias de resolución entre el viewport de OpenGL, la proyección usada, y la textura sobre la que

se renderiza. Para minimizar los problemas producidos al interpolar para adaptar los distintos tamaños, se cambia la proyección de modo que tenga el tamaño del área a renderizar, y se renderiza sobre una matriz de texturas (Tile Rendering). Aunque para imágenes extremadamente grandes y niveles de zoom extremos, siguen ocurriendo los problemas de interpolación.

12.6. Uniformidad del conjunto de normales calculado por CGAL

La aplicación utiliza la biblioteca CGAL para el cálculo de la geometría en operaciones CSG. Sin embargo, presenta un problema, las normales que proporciona no siempre presentan una dirección homogénea, produciendo resultados extraños al aplicar la iluminación. En la figura 12.1 se puede apreciar este hecho. Este problema no afecta a los sólidos primitivos, sólo a los compuestos, cuya geometría ha sido calculada por CGAL. Por el momento este problema no se ha podido solucionar.

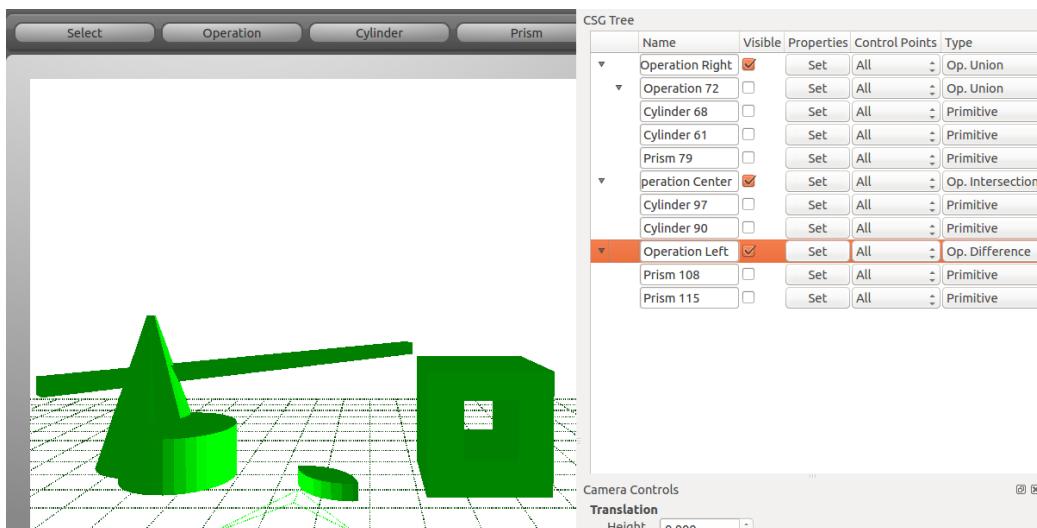


Figura 12.1: Problema en la iluminación de operaciones.

12.7. Concurrencia de contextos de OpenGL

Durante el desarrollo del proyecto surge la necesidad de mantener varios contextos activos de forma concurrente. Sin embargo debido a que OpenGL es una máquina de estados no reentrant, se requiere sincronizar las secciones de código que realizan llamadas de OpenGL, de forma que se indique a OpenGL cual es el contexto en el que se realizan.

Para ello, ya que se están usando los QGLWidget de Qt para crear ventanas que contengan un viewport de OpenGL y que manejan su propio contexto de OpenGL, se recurre a las órdenes `makeCurrent()` y `doneCurrent()` para indicar las secciones de código que pertenecen al contexto del QGLWidget que lo contiene. En general, todas las llamadas (aunque el código no sea parte de la clase QGLWidget de turno) son síncronas a métodos de los QGLWidgets excepto el cálculo de la geometría de las operaciones, que es síncrono al EntityTreeWidget (por controlar las interacciones entre operador y operandos) y esto, unido a la variabilidad de la geometría de los resultados de una operación booleana, fuerza a las operaciones a evitar reservar recursos en GPU y a trabajar usando recursos de CPU en exclusiva.

Capítulo 13

Mejoras y líneas de trabajo futuras

Ya que el proyecto es lo bastante amplio y ambicioso, aún quedan suficientes posibilidades de expansión y mejora. Por supuesto, hay una serie de prestaciones que, por falta de tiempo o por escalabilidad del proyecto con el objetivo de la defensa han ido quedándose sin implementar, pero no se han olvidado. Entre ellas se pueden destacar:

- Añadir más sólidos primitivos a la lista base (por ejemplo esferas, cuyo código existe ya en los puntos de control).
- Añadir las rotaciones sobre los ejes X y Z.
- Añadir la posibilidad de escalar sólidos.
- Crear un sistema de almacenaje y carga de los puntos generados por el algoritmo de punteado, de forma que complemente al sistema de almacenaje y carga de árboles CSG y permita guardar sesiones de trabajo sin necesidad de tener que recalcular nada.
- Optimizar las interacciones de forma que se permita trabajar a la vez con el motor 3D y el motor de stippling.
- Añadir herramientas inteligentes de modificación de los puntos generados en el proceso de punteado, de forma que se puedan aplicar de forma más directa los recursos de que normalmente dispone un artista al realizar el trabajo a mano.
- Mejorar la interfaz gráfica atendiendo a la usabilidad y experiencia de usuario.

- Incluir métodos de corrección de deformaciones de lente.
- Expandir la base de datos de cámaras e incluso crear un servicio para su distribución a través de internet.

Por supuesto, en una aplicación de este calibre, y con la limitación de recursos con que se han contado, es normal que queden algunos problemas por solventar. El problema que más limita la calidad de la aplicación es la necesidad de incrementar la precisión al interactuar con los sólidos, cambiando la escala logarítmica propia del z-buffer a una escala lineal más acorde a la precisión que se requiere en un programa de estas características.

Aún queda pendiente dar al usuario la opción de escoger la matriz de puntos de punteado (stipples), o incluso la implementación del algoritmo de generación de puntos descrito en [2]. Esto permitiría al usuario definir la resolución de los puntos usados y, por tanto, la resolución de la imagen final punteada. De hecho, es esta prestación la que permite al usuario decidir acerca del tamaño de la punta de la pluma virtual a usar. Este paso es particularmente importante para asegurar una impresión en papel correcta, de forma que no se desvirtúen las propiedades de los puntos. Una mejora más a incluir sería la de permitir al usuario escoger el tamaño de los puntos según la zona (es decir, según partes del modelo, tal como se ha hecho con las opciones de configuración específicas de punteado).

La generación de puntos se ha paralelizado y es independiente de la hebra principal de la interfaz gráfica de usuario, pero la carga de trabajo relacionada con la generación de la imagen final (renderización sobre archivo), queda pendiente de ser paralelizada. La razón por la que no se completado a tiempo, es el hecho de que para hacer la renderización sobre archivo, se necesita un contexto de OpenGL, y como ya se ha comentado en la sección de problemas y soluciones, debido a que OpenGL no es reentrant, la sincronización de hebras y contextos distintos resulta muy complicada y costosa en tiempo de desarrollo.

Una de las innovaciones introducidas en el algoritmo base es la capacidad de escoger el método de tramo. Esta capacidad nos permite poder hacer comparaciones de calidad de imágenes punteadas según el método de tramo escogido (y de hecho, la aplicación se ha diseñado de forma que añadir nuevos métodos sea fácil y rápido). Los métodos de tramo, no sólo se diferencian en lo bien que distribuyen el error de cuantización sin producir patrones de descarga, sino también en lo bien que modelan las siluetas y las

texturas, teniendo en cuenta también la eficiencia del mismo. La facilidad de implementación e inclusión de nuevos métodos de tramado, junto a la gran cantidad de métodos existentes, podría dar pie a una línea de investigación para determinar el método de tramado óptimo para aplicaciones de punteado mediante el uso de esta aplicación.

Otras mejoras que se pueden contemplar son:

- Hacer que la aplicación compile también bajo sistemas Mac.
- Permitir la carga de modelos 3D desde disco como primitivas (modelos específicos y muy detallados, sin componentes CSG, como por ejemplo, el modelo tridimensional de un dragón).
- Permitir inferir el modelo tridimensional a partir de un grupo de fotografías tomadas alrededor de el objeto en cuestión.
- Resolver el problema planteado en la sección 12.6.
- Permitir al usuario definir los parámetros de los métodos de detección de bordes para tener más control sobre la cantidad de bordes que se detectan.

Capítulo 14

Conclusiones

14.1. Conclusiones referentes a los objetivos

- Creación de una aplicación multiplataforma.

Este objetivo se ha alcanzado. La aplicación funciona sin problemas bajo sistemas Linux y Windows, y la única dificultad es la de instalar las bibliotecas. La utilización de CMake para dar soporte multiplataforma ha sido un gran acierto, y se podría incluso compilar bajo Mac sin realizar muchos cambios.

- Interfaz gráfica de usuario orientada a ser usada por artistas.

En principio este objetivo se ha cumplido, pero debido a la naturaleza subjetiva de los requerimientos, se debería realizar un análisis de usabilidad centrado en artistas, para revelar imperfecciones y mejorar la interfaz.

- Reconstrucción de escena.

Este objetivo se ha alcanzado, y en el estado en que se encuentra la aplicación, es capaz de calcular el campo de visión y estimar la distancia entre la cámara y el objetivo fotografiado requiriendo una mínima interacción con el usuario. Sólo en el caso de que la imagen no disponga de los metadatos necesarios se requerirá una interacción especial por parte del usuario.

- Motor 3D con capacidad de creación de modelos complejos e iluminación.

La aplicación cumple los objetivos propuestos a este respecto, si bien no está exenta de espacio para su mejora. La aplicación es capaz de crear y editar modelos simples, pero la variedad es aún demasiado pequeña para ser usada con suficiente eficiencia. De hecho, se considera la posibilidad de mejorar el sistema utilizando técnicas más avanzadas.

La aplicación es capaz de mantener el árbol CSG y las operaciones se calculan perfectamente. En cuanto a las interacciones con la posición, orientación y geometría del modelo, funcionan correctamente, si bien se debería añadir un modo de precisión para poder maximizar la calidad de los resultados. El resto de los objetivos dentro de esta sección se cumplen excepto por el problema presentado en la sección 12.6 que queda pendiente de ser solucionado.

- Mejora de los algoritmos de punteado mediante el uso de información tridimensional.

Los objetivos relacionados con esta sección se han alcanzado, y se logra obtener una imagen punteada dependiente del modelo tridimensional y de las configuraciones específicas de sus partes. La imagen punteada final se almacena en disco en un único fichero, pero se puede modificar el código con mucha sencillez en caso de necesitarse una salida a trozos (tiles). La aplicación produce imágenes punteadas de mayor resolución que la entrada proporcionada, pero aún no permite al usuario escoger el conjunto de puntos a usar (aunque soporta múltiples conjuntos de puntos, pudiendo cambiarse de forma sencilla recompilando).

14.2. Valoración personal

Tras haber invertido tanto tiempo y esfuerzo en este proyecto, es ahora cuando puedo contemplarlo en retrospectiva. Lo cierto es que al principio el proyecto me parecía similar al resto de proyectos que se me habían propuesto hasta la fecha, pero ahora puedo ver la ambición y el potencial que posee. Con esto quiero decir que el proyecto no acaba aquí, sino que sigue y seguirá creciendo, mejorando. De hecho durante toda la etapa de desarrollo,

se han cambiado requerimientos e incluso se han añadido otros favoreciendo la innovación y el crecimiento del mismo. Por ejemplo, la inocente idea que tuve de implementar otro algoritmo de tramado, pues cuando implementé Floyd-Steinberg no me complacieron los resultados, ha permitido mejorar el resultado, y creo que merece la pena experimentar también con otros algoritmos más avanzados.

Desde un primer momento, una de las dificultades más grandes que he enfrentado era el tamaño del equipo de desarrollo. Aunque he contado con el apoyo de los equipos del LRV y Virtum Graphics, y con Germán (que ha aportado también un par de clases al proyecto), en general desarrollaba solo. Haber trabajado así me ha hecho apreciar aún más las bondades del trabajo en equipo, y siempre que he podido, he intentado incluir a otros en los procesos de diseño para minimizar el efecto que produce el tener un único punto de vista (me refiero, por supuesto a las decisiones de diseño que sólo me concernían a mí, pues el resto se discutían en las reuniones).

En general, este proyecto, no sólo ha sido muy productivo desde el punto de vista formativo, sino que también espero que suponga un firme primer paso en mi carrera profesional. De hecho, yo veo más el proyecto como un prototipo funcional, que como un producto acabado o casi acabado. Esto es así debido al potencial de mejora que demuestra y a la gran cantidad de ideas que se pueden aplicar para obtener una aplicación de punteado de gran calidad, que incluso pueda tener salida comercial.

Bibliografía

- [1] ANAND, V. B. *Computer graphics and geometric modeling for engineers*. John Wiley & Sons, Inc., 1996.
- [2] ARROYO, G., MARTÍN, D., AND LUZÓN, M. V. Stochastic generation of dots for computer aided stippling. *Computer-Aided Design and Applications* 7, 4 (2010), 447–463.
- [3] BLANCHETTE, J., AND SUMMERFIELD, M. *C++ GUI Programming With Qt 4* (*Prentice Hall Open Source Software Development Series*) *Author: Jasmin Blanch*. Prentice Hall, 2008.
- [4] BUSCH, D. *David Busch's Canon EOS Rebel T3i/600D Guide to Digital SLR Photography*. David Busch Camera Guides. Course Technology Ptr, 2011.
- [5] DALAL, K., KLEIN, A. W., LIU, Y., AND SMITH, K. A spectral approach to npr packing. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2006), NPAR '06, ACM, pp. 71–78.
- [6] DEUSSEN, O., HILLER, S., VAN OVERVELD, C., AND STROTHOTTE, T. Floating points: A method for computing stipple drawings. *Computer Graphics Forum* 19, 3 (2000), 41–50.
- [7] FOSTER, K., JEPP, P., WYVILL, B., SOUSA, M., GALBRAITH, C., AND JORGE, J. Pen-and-ink for blobtree implicit models. *Computer Graphics Forum* 24, 3 (2005), 267–276.
- [8] GOOCH, B., AND GOOCH, A. *Non-photorealistic rendering*. Ak Peters Series. A K Peters, 2001.
- [9] HEARN, D., AND BAKER, M. P. *Computer graphics, C version*, vol. 2. Prentice Hall Upper Saddle River, 1997.

- [10] HECHT, E., AND ZAJĄC, A. *Optics*. Physics Series. Addison-Wesley Publishing Company, 1987.
- [11] HERTZMANN, A. A survey of stroke-based rendering. *Computer Graphics and Applications, IEEE* 23, 4 (July 2003), 70–81.
- [12] ISENBERG, T., NEUMANN, P., CARPENDALE, S., SOUSA, M. C., AND JORGE, J. A. Non-photorealistic rendering in context: An observational study. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2006), NPAR ’06, ACM, pp. 115–126.
- [13] JAHANSHAHI, M. R., AND MASRI, S. F. A new methodology for non-contact accurate crack width measurement through photogrammetry for automated structural safety evaluation. *Smart Materials and Structures* 22, 3 (2013), 035019.
- [14] KIM, D., SON, M., LEE, Y., KANG, H., AND LEE, S. Feature-guided image stippling. *Computer Graphics Forum* 27, 4 (2008), 1209–1216.
- [15] KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. Recursive wang tiles for real-time blue noise. *ACM Trans. Graph.* 25, 3 (July 2006), 509–518.
- [16] LLOYD, S. Least squares quantization in pcm. *Information Theory, IEEE Transactions on* 28, 2 (Mar 1982), 129–137.
- [17] LU, A., MORRIS, C. J., EBERT, D., RHEINGANS, P., AND HANSEN, C. Non-photorealistic volume rendering using stippling techniques. *Visualization Conference, IEEE* 0 (2002), null.
- [18] LU, A., MORRIS, C. J., TAYLOR, J., EBERT, D. S., HANSEN, C. D., RHEINGANS, P., AND HARTNER, M. Illustrative interactive stipple rendering. *IEEE Trans. Vis. Comput. Graph.* 9, 2 (2003), 127–138.
- [19] LU, A., TAYLOR, J., HARTNER, M., EBERT, D. S., AND HANSEN, C. D. Hardware-Accelerated Interactive Illustrative Stipple Drawing of Polygonal Objects. In *Proceedings of Vision, Modeling, and Visualization 2002 (VMV 2002, November 20–22, 2002, Erlangen, Germany)* (Berlin, 2002), B. Girod, H. Niemann, H.-P. Seidel, G. Greiner, and T. Ertl, Eds., Akademische Verlagsgesellschaft Aka GmbH, pp. 61–68.
- [20] MARTÍN, D., ARROYO, G., LUZÓN, M. V., AND ISENBERG, T. Example-based stippling using a scale-dependent grayscale process. In

Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering (New York, NY, USA, 2010), NPAR '10, ACM, pp. 51–61.

- [21] MARTÍN, D., ARROYO, G., LUZÓN, M. V., AND ISENBERG, T. Scale-dependent and example-based grayscale stippling. *Computers & Graphics* 35, 1 (2011), 160 – 174. Extended Papers from Non-Photorealistic Animation and Rendering (NPAR) 2010.
- [22] MCCOOL, M., AND FIUME, E. Hierarchical poisson disk sampling distributions. In *Proceedings of the Conference on Graphics Interface '92* (San Francisco, CA, USA, 1992), Morgan Kaufmann Publishers Inc., pp. 94–105.
- [23] MOULD, D. Stipple placement using distance in a weighted graph. In *Proceedings of the Third Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging* (Aire-la-Ville, Switzerland, Switzerland, 2007), Computational Aesthetics'07, Eurographics Association, pp. 45–52.
- [24] PASTOR, O. M., FREUDENBERG, B., AND STROTHOTTE, T. Real-time animated stippling. *IEEE Computer Graphics and Applications* 23, 4 (2003), 62–68.
- [25] SCHLECHTWEG, S., GERMER, T., AND STROTHOTTE, T. Renderbots—multi-agent systems for direct image generation. *Computer Graphics Forum* 24, 2 (2005), 137–148.
- [26] SCHMIDT, R., ISENBERG, T., JEPP, P., SINGH, K., AND WYVILL, B. Sketching, scaffolding, and inking: A visual history for interactive 3d modeling. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2007), NPAR '07, ACM, pp. 23–32.
- [27] SCHWABER, K., AND BEEDLE, M. *Agile Software Development with Scrum*, 1st ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
- [28] SECORD, A. Weighted voronoi stippling. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2002), NPAR '02, ACM, pp. 37–43.
- [29] SECORD, A., HEIDRICH, W., AND STREIT, L. Fast primitive distribution for illustration. In *Proceedings of the 13th Eurographics Workshop*

on Rendering (Aire-la-Ville, Switzerland, Switzerland, 2002), EGRW '02, Eurographics Association, pp. 215–226.

- [30] SHREINER, D., GROUP, B. T. K. O. A. W., ET AL. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009.
- [31] SOUSA, M. C., FOSTER, K., WYVILL, B., AND SAMAVATI, F. Precise ink drawing of 3d models. *Computer Graphics Forum* 22, 3 (2003), 369–379.
- [32] THELIN, J. *Foundations of Qt development*, vol. 7. Springer, 2007.
- [33] VANDERHAEGHE, D., BARLA, P., THOLLOT, J., AND SILLION, F. Dynamic point distribution for stroke-based rendering. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)* (2007), pp. 139–146.
- [34] VINCE, J. A. *Geometric algebra for computer graphics*, vol. 1. Springer, 2008.
- [35] WINKENBACH, G., AND SALESIN, D. H. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 91–100.
- [36] WOO, M., NEIDER, J., DAVIS, T., AND SHREINER, D. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*, 3rd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [37] WRIGHT, R. S., HAEMEL, N., SELLERS, G. M., AND LIPCHAK, B. *OpenGL SuperBible: comprehensive tutorial and reference*. Pearson Education, 2010.
- [38] YUAN, X., NGUYEN, M. X., ZHANG, N., AND CHEN, B. Stippling and silhouettes rendering in geometry-image space. In *Proceedings of Eurographics Symposium on Rendering (EGSR'05)* (2005), the Eurographics Association, pp. 193–200, color plate 318.

Apéndice A

Recorridos de árboles

Debido al uso en el proyecto de estructuras de tipo árbol, ha sido necesario incluir recorridos sobre árboles.

A.1. Primero en anchura

Se recorren los nodos por niveles, de izquierda a derecha y desde la raíz a las hojas.

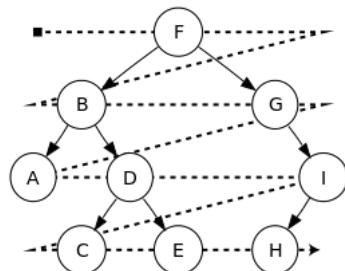


Figura A.1: Recorrido primero en anchura (fuente: Wikipedia).

A.2. Primero en profundidad

Se ha usado el recorrido pre-orden. Se recorren los nodos profundizando primero a través de la rama izquierda, para luego volver atrás y profundizar por la siguiente rama.

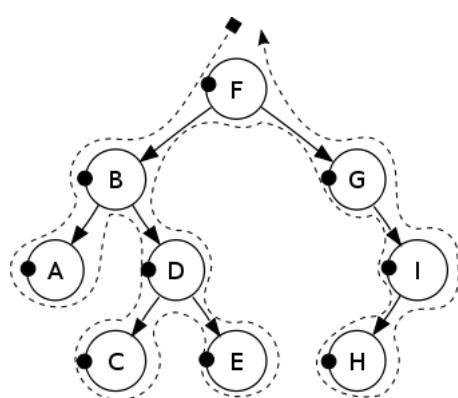


Figura A.2: Recorrido primero en profundidad, pre orden (fuente: Wikipedia).

Apéndice B

Umbralización

La umbralización (thresholding) es el método más sencillo de segmentación. Consiste en, a partir de una imagen en escala de grises, transformarla en una imagen binaria (dos niveles, blanco y negro). El algoritmo es realmente sencillo, se establece un umbral contra el cual se compara el nivel de gris de cada pixel de la imagen, y según el resultado, se establece el valor del pixel a un valor máximo o mínimo. En el proyecto se usa como parte de los algoritmos de tramado, como componente principal al cuantizar la imagen.



Figura B.1: Original



Figura B.2: Umbralizada

Apéndice C

Tramado

El tramado (dithering) es un método de reducción de control consistente en la introducción intencionada de ruido para distribuir el error de cuantificación de forma que evite la aparición de patrones erróneos tales como el «color banding».

Los múltiples algoritmos de tramado comparten una misma estructura. Se recorre la imagen pixel a pixel, fila a fila, en orden (de izquierda a derecha y de arriba a abajo). Se umbraliza el pixel, y el error de cuantificación se distribuye entre píxeles de la vecindad (píxeles aún no procesados).

El filtro usado para distribuir el error de cuantificación es lo que caracteriza y diferencia. Por supuesto, cuanto mayor es el filtro, menos eficiente será el método, pero mejor distribuirá el error de cuantificación, y mejor será el resultado.



Figura C.1: Original



Figura C.2: Floyd-Steinberg



Figura C.3: Stucki

Como se puede observar en la comparación expuesta en las figuras C.1, C.2 y C.3, el algoritmo de Stucki produce los mejores resultados, aunque no está exento los problemas que experimenta en mayor medida el algoritmo de Floyd-Steinberg. Dichos problemas son los patrones de descarga de error de cuantificación, que se acumulan en zonas, creando bloques oscuros que nada tienen que ver con la imagen original. Estos patrones son especialmente apreciables en la zona del hombro de la estatua.

Aunque ninguno de los dos algoritmos produce un resultado perfecto, el algoritmo de Stucki produce un resultado más preciso, con siluetas mejor definidas, si bien también requiere de un mayor tiempo de computación. En caso de que el tiempo de computación no suponga problema, merece la pena considerar algoritmos de trazado más avanzados tales como Sierra, Two-Row Sierra o Sierra Lite.

C.1. Floyd-Steinberg

El filtro usado en el método de Floyd-Steinberg es el siguiente:

$$\begin{pmatrix} & X & 7/16 \\ 3/16 & 5/16 & 1/16 \end{pmatrix}$$

La X representa el pixel sobre el que se aplica el filtro. Como se puede observar, el filtro no se aplica sobre píxeles que ya se han recorrido, y debido al tamaño del filtro, es necesario añadir un borde artificial de tamaño de un pixel alrededor de la imagen.

C.2. Stucki

El filtro usado en el método de Stucki es el siguiente:

$$\begin{pmatrix} & & & & \\ & X & 8/42 & 4/42 & \\ 2/42 & 4/42 & 8/42 & 4/42 & 2/42 \\ 1/42 & 2/42 & 4/42 & 2/42 & 1/42 \end{pmatrix}$$

La X representa el pixel sobre el que se aplica el filtro. Como se puede observar, el filtro no se aplica sobre píxeles que ya se han recorrido, y debido al tamaño del filtro, es necesario añadir un borde artificial de tamaño de dos pixeles alrededor de la imagen.

Apéndice D

Código ajeno, modificado o incluido

D.1. Frame Buffer Objects

Con el objetivo de acelerar el desarrollo del proyecto, y para facilitar mi aprendizaje, Germán tuvo la amabilidad de permitirme usar una clase que él desarrolló para la creación y gestión de los Frame Buffer Objects (FBOs) que se usan de forma extensiva en el proyecto. La clase proporcionada es la clase S3DFBO, que se define en los ficheros framebuffer.hh y framebuffer.cc. Esta clase sólo se ha incluido en el proyecto y no se ha modificado.

Por defecto OpenGL utiliza un único Frame Buffer, pero hay situaciones en que es interesante disponer de otros Frame Buffers sobre los que renderizar. Es esta la razón de existir de los Frame Buffer Objects, que OpenGL pone a disposición del programador. Gracias a la clase S3DFBO, el uso y mantenimiento de un FBO se simplifica sobremanera. A continuación se incluye un ejemplo comentado de uso.

```
// Create the FBO with the desired size
unsigned int width = 500;
unsigned int height = 500;
S3DFBO * fbo = new S3DFBO( width , height ,
                           GL_RGB,GL_RGB,GL_FLOAT,GL_NEAREST,
                           true ,true );

// Now rendering on fbo
fbo->renderFBO();
```

```

// Insert Rendering code here...
fbo->renderFrameBuffer();
// Now rendering back on the default Frame Buffer

// Get the textureID that holds what was rendered
// before, it's ready to be used.
GLuint textureID = fbo->getTexture();

// Once it's not needed anymore, free the resources...
delete fbo;
fbo = 0;

```

D.2. Soporte de la biblioteca CGAL

Debido a que el proyecto requiere dar soporte a la creación y mantenimiento de un árbol CSG (Constructive Solid Geometry). Para ello Germán contribuyó al proyecto con un par de clases para definir superficies y estructuras tridimensionales booleanas mediante la biblioteca CGAL. Dichas clases, SurfaceM y BoolMesh, que están definidas en los ficheros boolmesh.h y boolmesh.cpp. Ya que el árbol CSG forma parte del núcleo de la aplicación, se procedió a sobrecargar algunos métodos para facilitar su uso. También se corrigió un bug presente en uno de los métodos. A continuación se explicitan los cambios:

- En la clase SurfaceM, se sobrecarga el método addVertex para que soporte vectores glm::vec3.
- En la clase SurfaceM, se sobrecarga el método addFace para que admita
- En la clase BoolMesh, se cambian los nombres de las constantes que designan las operaciones booleanas por coincidencia con otras variables definidas previamente por Qt.
- En la clase BoolMesh, se sobrecarga el método multMatrix para que soporte matrices glm::mat4.
- En la clase BoolMesh, se corrige el método verticesCache(), ya que sufría un par de errores (inicialización incorrecta del tamaño del array en memoria e iterador incorrecto) que provocaban errores de segmentación.

Debido a la extensión que necesitaría un ejemplo de uso, se procede a explicar en términos generales su funcionamiento.

El primer paso es crear una superficie (Surface), y añadir, primero los vértices de la figura geométrica a representar, y luego las caras (usando los índices de los vértices, según el orden de adicción). Una vez la superficie se ha definido, se crea un objeto de tipo BoolMesh, se limpia la caché mediante el método clearCache(), se añade la superficie mediante el método addSurface(), y finalmente se llama al método flushCache(). Finalmente, mediante los métodos verticesCache() y facesCache() se obtiene la geometría calculada por CGAL.

Por supuesto, el sistema explicado sirve para definir sólidos primitivos, pero una vez definidos, y usando los métodos multMatrix() (para aplicar las transformaciones de modelo) y boolean() (para calcular la operación booleana pasada como argumento), se obtiene un nuevo BoolMesh con la geometría resultante, que es extraída de la misma manera que con los sólidos primitivos mencionados anteriormente.

CGAL es una biblioteca de gran potencia, que ha permitido incluir y mantener un árbol CSG en la aplicación, con una inversión mínima de tiempo, y que de no haber sido incluida, habría provocado importantes retrasos en el proyecto.