Knight Lore 20XX: utilizando tcnicas de grficos por ordenador para llevar un juego clsico a la tecnologa moderna

Ricard Galvany[1] and Gustavo Patow[1]

ViRVIG-UdG, Spain

Resumen En muchos aspectos, los juegos de ordenador desarrollados durante la década de los 80 son una parte fundamental de la historia de los juegos modernos, ya que establecieron las bases de lo que actualmente entendemos por entretenimiento electrónico. Sin embargo, la tecnología utilizada para esos juegos se han vuelto obsoleta. A pesar de ello, cuando se juega de nuevo a unos de estos clásicos, probablemente a través de un emulador, la experiencia no está a la par con los estándares modernos. Por esta razón hemos decidido desarrollar Knight Lore 20XX, como un experimento de la utilización técnicas de informática gráfica para traer un juego clásico de los años 80 a la tecnología moderna. Este artículo busca informar acerca de la experiencia, los problemas encontrados y las soluciones desarrolladas. Mientras que la naturaleza simple del juego original prevé una cierta estética en sí misma, creemos que nuestro método puede producir un juego en 3D convincente que captura de alguna forma el encanto del original, y que podría ser utilizado para actualizar cualquier otro juego clásico.

1. Introducción

El objetivo de este proyecto fue desarrollar el remake de un juego clásico del *ZX-Spectrum* llamado *Knight Lore*. Para conseguir resultados lo más cercanos al juego original, se ha optado por utilizar un nuevo enfoque donde, en lugar de reconstruir de nuevo el juego desde cero, queremos aprovechar al máximo el original. Dada que éste funciona sobre los desaparecidos ordenadores Spectrum, nos vemos forzados a trabajar sobre un emulador. La idea principal fue realizar la ingeniería inversa del juego para posteriormente desviar el flujo de datos en el momento de generar las imágenes, utilizando nuestros algoritmos 3D para crear la escena a partir de los datos generados por el propio juego. De esta forma la jugabilidad permanecerá intacta, sólo actualizaremos los gráficos. El motivo principal para escoger el juego Knight Lore es que fue realizado con una técnica pseudo 3D (llamada técnica *Filmation* [2]), que permitie una reconstrucción parcial de la escena.

2. Antecedentes

Pixel Art es una forma de arte digital donde los detalles en la imagen son representado a nivel de píxel. Los gráficos en prácticamente todos los ordenadores y juegos de vídeo antes de mediados de la década de los 90 consistían, sobre todo, en imágenes Pixel Art. Estos videojuegos siguen siendo disfrutado hoy en día, gracias a los numerosos emuladores que se han desarrollado para reemplazar el hardware que se volvió obsoleto hace mucho tiempo. En dicha técnica, el hecho de que cada píxel fuese colocado manualmente permitía alcanzar un máximo de expresión y significado por cada píxel. Kopf y Lichinsky [1] propusieron un método para producir arte vectorial convincente a partir de imágenes en Pixel Art generadas con un emulador, de manera que su algoritmo se las arregla para capturar algunas de las características y preservar el encanto del original. Otro ejemplo es el trabajo de Thibeault [4], que busca reconocer elementos de pixel art, reemplazándolos dinámicamente por imágenes de nueva creación. La escena retro"también ha desarrollado interesantes técnicas con el mismo objetivo, como los packs de texturas de alta resolución desarrollados para la emulación de la Nintendo 64 [3].

3. KnightLore

Knight Lore fue un juego creado por la empresa *Ultimate Play The Game* en 1984. El videojuego fue desarrollado por Tim Stamper y Chris Stamper para ZX Spectrum, Amstrad CPC y MSX. En él, el jugador debe conseguir los ingredientes de una poción repartidos en un castillo lleno de trampas y entregárselos a un mago para que lo libre de la maldición que transforma al personaje en lobo por la noche. El juego se trata de un clásico juego de plataformas, su principal novedad reside en el entorno gráfico, que utiliza perspectiva isométrica que permite no sólo moverse en tres dimensiones, sino también interaccionar con objetos mediante una física sencilla pero efectiva. El motor utilizado llevó el nombre de *Filmation* y se convirtió en la marca principal de la casa.

3.1. Datos estáticos del Knight Lore

Mapa: Los lugares de juego se juega en una cuadrícula de 16×16. Ver Figura 1, izquierda. La dirección de movimiento se calcula de forma implícita mediante la adición al código de ubicación de un valor fijo dependiendo de la dirección (16 norte, -16 sur, -1 oeste, 1 este). Ver Figura 1, derecha. Los datos de las habitaciones (sólo las utilizadas) se almacenan en una

lista secuencial de habitaciones. Para localizar una habitación se debe realizar una búsqueda iterativa por dicha lista.

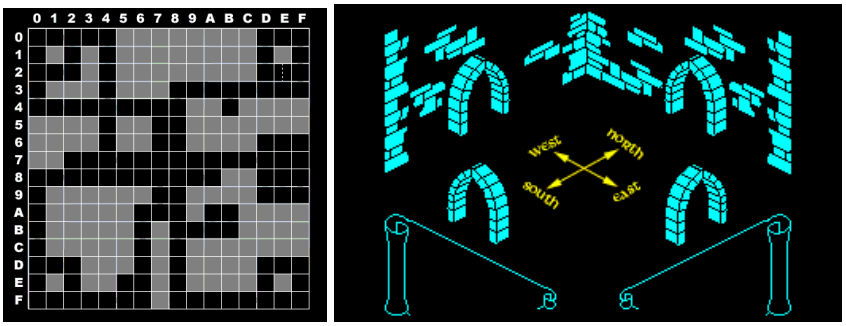

Figura 1. Izquierda: Mapa del juego. Derecha: Orientaciones en el juego.

Habitaciones: Para cada entrada de la lista de habitaciones, el primer byte representa el identificador (número) y el segundo es el tamaño de los datos de la habitación. A continuación, cada entrada almacena otros datos como las dimensiones y el color, el código de las paredes, y un listado de los objetos contenidos en la habitación. Ver Figura 2.

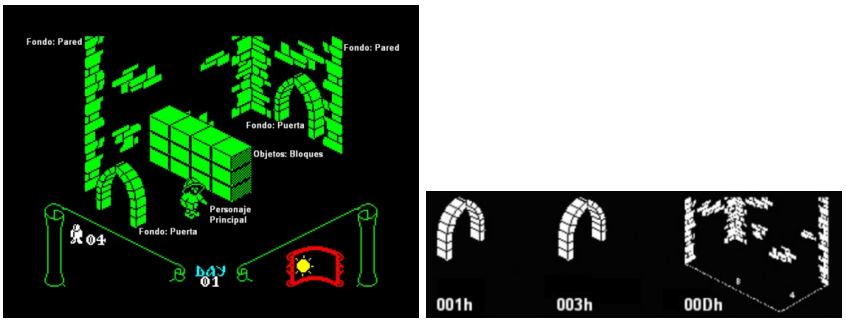

Figura 2. Izquierda: Pantalla del juego con sus elementos identificados. Derecha: Fondos de una habitación ($0Eh$)

Fondos A continuación, se pasa a leer los datos de los fondos, donde cada byte representa el identificador de un fondo hasta que aparezca uno con valor FFh. El byte con ese valor simplemente determina que empieza la sección de los objetos.

Objetos: Por último, los datos de objetos están codificados en una lista que contiene, para cada uno, su identificador y número de instancias, seguido por la información de las posiciones de cada una. Este resultado puede verse en la figura 3, izquierda. El estado de la misma habitación, unos segundos más tarde, se encuentra en la parte derecha de la misma figura, lo que nos permite ver que el estado dinámico no puede capturarse de estos datos.

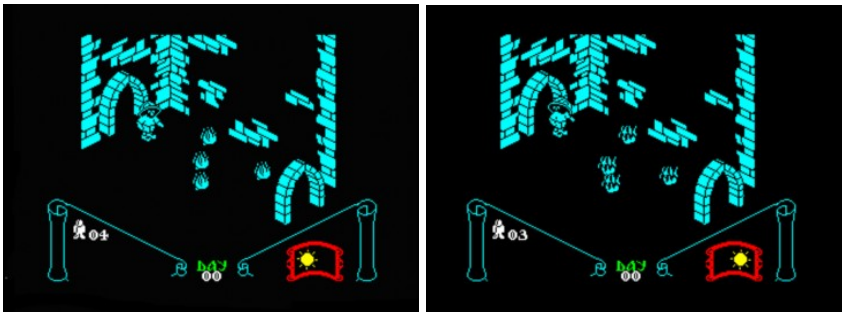

Figura 3. Una habitación al momento de ser inicializada y unos segundos desués.

3.2. Habitaciones en la memoria de trabajo

El juego utiliza un espacio de memoria llamado memoria de trabajo (*scratch mem*, que está situada entre las direcciones de memoria $5BA0h$ y $6107h$) y es la que refleja los datos dinámicos de las localizaciones. Estudiando el código fuente del juego desensamblado, observamos la función llamada *RetrieveScreen*. Dicha función, básciamente, itera sobre la lista de habitaciones buscando la que se necesita en cada momento, y descomprime sus datos de localización, fondo y objetos en memoria de trabajo.

Fondos: RetrieveScreen obtiene los fondos de las habitaciones indexando el código obtenido en una tabla alamacenada de forma estática a partir de la dirección $6CE2h$, que permite obtener la posición de los datos de un fondo simplemente añadiendo a una dirección base el valor de cada fondo. Ver Figura 4. Los datos de esta forma indexados nos dan el ID del sprite ultilizado, sus coordenadas (x, y, z) en el mundo, así como su ancho, alto y largo.

Figura 4. Sprites en el juego. formación de la imagen del arco este. Se puede observar claramente como la imagen final es el resultado de componer dos sprites más simples.

Objetos: A continuación, la función *RetrieveScreen* itera sobre los objetos en la habitación para recuperar su información 3D y su representación en espacio de imagen. Para ello se utiliza otra tabla de indirecciones posicionada en $6BD1h$, y se le suma el ID del objeto a escoger multiplicado por 2 (la cantidad de bytes que ocupa una dirección de memoria en el ZX Spectrum). Por ejemplo, el fuego tiene identificador $0Ah$, lo que nos conduce a la posición $6BE9h$, donde obtendremos las coordenadas x, y, z, así como una referencia a su sprite y si debe dibujarse invertido (modo *espejo*) o no: cabe recordar que el ZXSpectrum era un ordenador muy limitado en cuanto a memoria. Hay un serie de objetos (los que se mueven de manera regular, como por ejemplo los guardias o el fuego) que requieren cambioar de posición en función del movimiento que realizan. Esta información también se encuentra especificada en estos bytes, que básicamente son flags que determinan si alguna de las coordenadas requiere de la corrección. Cada instancia de un objeto tendrá una entrada independiente en la zona de trabajo.

Resolución 3D alta y baja: Aunque las coordenados de los fondos, objetos y el personaje principal inicialmente estuvieran en 3D de baja resolución (low), en la memoria de trabajo se almacenaban en 3D de alta resolución (high). Utilizando el siguiente sistema de ecuaciones, podemos pasar de 3D Low a 3D High y viceversa, que era lo que nos interesaba, pues la información dinámica trabajaba en 3D de alta resolución [5]:

$$\begin{aligned} X_H &= 16X_L + 8X_s + 72 \\ Y_H &= 16Y_L + 8Y_s + 72 \\ Z_H &= 12Z_L + 4Z_s + ScreenZ \end{aligned}$$

dónde X_L, Y_L, Z_L son las coordenadas en 3D de baja resolución (leídas de la información estática del juego), X_s, Y_s, Z_s son valores que forman

parte de la información que define los sprites y que permiten trabajar con los objetos que se desplazan solos. En función de cada objeto tienen un valor u otro. Finalmente $ScreenZ$ es una constante que siempre vale 128. Ver la Figura 5.

Figura 5. Resultado de cambiar las coordenadas de baja resolución de un objeto.

Personaje Principal: Para obtener información de cómo se almacena al personaje principal, realizamos una comparación de la memoria de trabajo variando sólo la posición del mismo y dejando fijos todos los otros parámetros (anulando las respectivas funciones de evolución). Ver la Figura 6, izquierda. Las coordenadas del personaje están separadas en dos bloques: las situadas en las posiciones de memoria $5C09h$, $5C0Ah$ y $5C0Bh$ representan las coordenadas x, y, z de la posición cuerpo del personaje principal, y las situadas en las posiciones de memoria $5C20h$, $5C2Ah$ y $5C2Bh$, representa la posición de la cabeza. Ver Figura 6, derecha. La representación de los personajes en el juego en general está formada por dos bloques, para así poder aplicar diferentes animaciones de manera separada. Para el personaje principal, los bytes situados en la direcciones $5C41h$ y $5C45h$ indican los sprites del personaje a utilizar. Para acabar de determinar la orientación, se utilizan los bytes de las direcciones $5C0Fh$ y $5C2Fh$, que sirven para diferenciar los sprites que tienen el mismo identificador pero que el motor gráfico invierte, aplicando una operación de espejo.

Figura 6. Coordenadas (izquierda)y sprites (derecha) que forman el personaje principal.

4. Implementación del Knight Lore 20XX

Llegado este punto nos surgió una problemática al pasar el control al nuevo entorno 3D. Por definición, OpenGL es una máquina de estados, y una vez se le ha pasado el control, itera indefinidamente hasta que se termina el programa o se le obliga a terminar. Esto hacía que el emulador se quedara "estancado" ejecutando el entorno 3D. Para ello, en lugar de ejecutar la función que realiza una iteración del emulador, y luego ejecutar la que genera el entorno 3D, se optó por conceder el control absoluto al entorno 3D y mediante funciones propias de OpenGL que permiten especificar qué hacer en períodos de inactividad, ir concediendo iteraciones a la función de emulación. Vemos que estas dos disposiciones hacen que se vaya dibujando el nuevo entorno cuando es necesario, y a la vez se ejecuta el código de emulador.

Para adquirir los datos del juego, primero se creó una función que adquiriera toda la información geométrica que se requiriese para la recreación del juego, a partir de la información almacenada en la información inicial estática que define las localizaciones, la memoria de trabajo, la tabla de los datos de los fondos, y labla de los datos de los objetos.

Fondos Las paredes de cada habitación se dibujaron utilizando las texturas del juego original. El proceso comprendió la captura de cada sprite (ver Figura 7), que conformaba cada pared para su posterior almacenamiento en un fichero gráfico. La función de dibujado de fondos examina los datos dinámicos de la habitación, para obtener los identificadores de los sprites que formarán parte de la pared. Después, carga los ficheros que corresponden a los sprites de la pared y los coloca en función de su posición en el nuevo entorno. Hay que tener en cuenta que las texturas capturadas del juego original presentan una distorsión en perspectiva ortográfica, pero que se solucionó gracias a que, mediante OpenGL, a la hora de aplicar una textura a un objeto, podemos corregir su posicionamiento

mediante la técnica llamada *dewarping*. La técnica dewarping elige una serie de puntos, que se pasan como coordenadas de textura a OpenGL. Entonces éste último realiza las correcciones necesarias para aplicar la textura para su correcta visualización. Conceptualmente se muestra en la Figura 7. El resto de los fondos (mayoritariamente puertas) se dibujaron como cualquier otro objeto, utilizando geometría modelada al efecto.

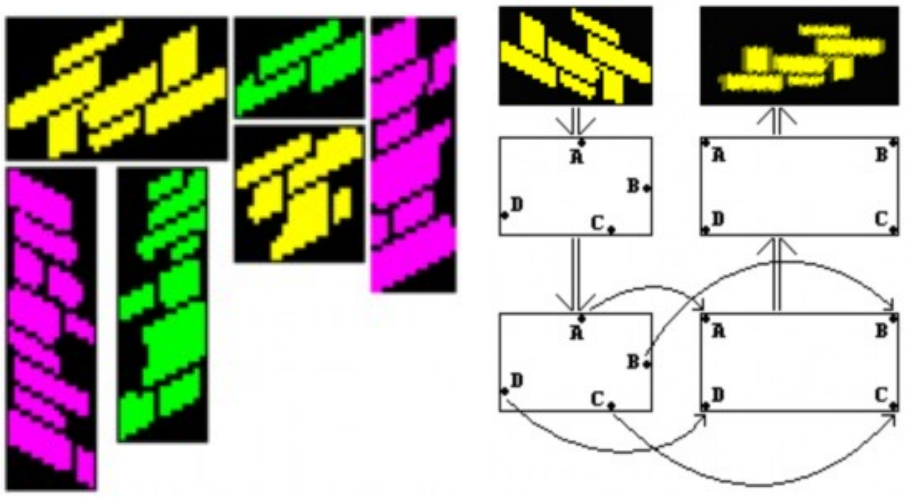

Figura 7. Izquierda: Diferentes sprites para las paredes. Derecha: la técnica dewarping.

Objetos En nuestro sistema, todos los objetos son dibujados mediante primitivas OpenGl. Ver la Figura [**?**]. La función que se encarga de dibujarlos es llamada por la función de principal tantas veces como objetos dispone la localización. Cada llamada incluye los argumentos necesarios que describen el objeto. En función del identificador que recibe, muestra unas primitivas u otras. Si hay algún objeto no definido, entonces se dibuja el objeto por defecto, que en nuestro caso es una pequeña esfera.

Hay que destacar que, aunque hay varios objetos que comparten un aspecto visual, de caras a la interacción en el juego se comportan de diferente manera, e incluso no disponen del mismo identificador de sprite. Es el caso de un objeto muy común, el bloque, el cual dispone de diferentes entidades (bloque fijo, bloque que desaparece al ser pisado, bloque con movimiento), que internamente el juego trata de manera diferente, pero visualmente se representan igual. Los objetos que disponen de animación (pelota, fuego, soldado, etc.) requieren de diferentes sprites para repre-

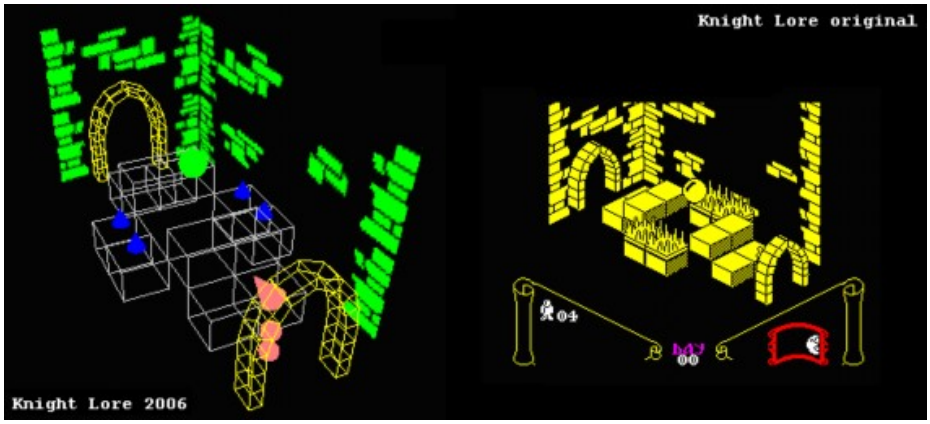

Figura 8. Ejemplo de la visualización resultante con nuestro algoritmo.

sentarla. También en este caso es algo transparente para el nuevo entorno del juego, ya que el juego original va modificando la memoria de trabajo en función del sprite que requiere cada objeto animado y de su estado.

Personaje Principal La función que dibuja el personaje principal, primeramente obtiene las coordenadas (x, y, z) situadas en las posiciones de memoria $5C09h$, $5C0Ah$ y $5C0Bh$ (ver sección 3.2). Seguidamente convierte estos valores de coordenadas 3D High al entorno OpenGL. Una vez conocida la posición, se debe deducir el estado del personaje (hombre o lobo) y su orientación. Consultando la posición de memoria $5C41h$ obtendremos el identificador del sprite del cuerpo del personaje, y con ello su estado y orientación. Para terminar de saberla, deberemos consultar la posición de memoria $5C0Fh$, que nos dice si se debe hacer una operación de espejo o no. Una vez adquiridos los datos del personaje para dibujarlo, aplicamos una rotación, posteriormente una traslación y luego utilizamos varias primitivas de OpenGL para finalmente dibujar el personaje. Ver Figura 9.

5. Resultados

El resultado final consta de un nuevo entorno, como podemos observar en la figura 10, abajo. La figura 10, arriba, muestra una secuencia del movimiento de un balón botando. También dispone de la opción de visualizar el juego en 3D desde diferentes perspectivas y un mejorado aspecto visual.

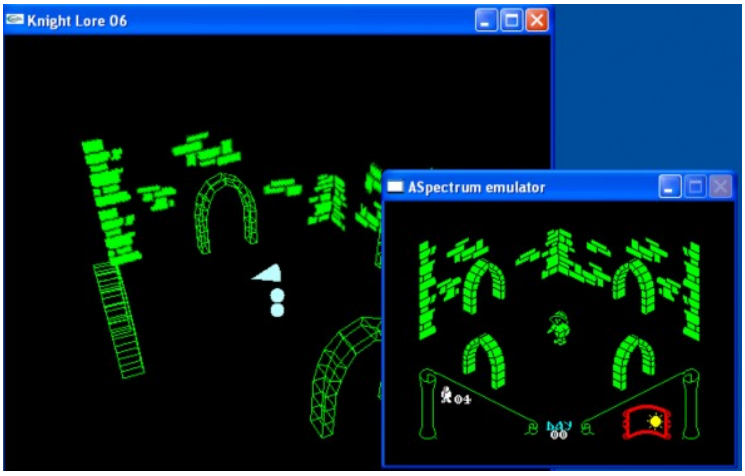

Figura 9. Personaje principal en el viejo y nuevo contexto.

Dadas las características del juego original, su ejecución se realiza completamente en un emulador de ZXSpectrum, lo cual representa un costo realmente bajo para los estándares actuales. Por lo tanto, el ordenador dispone de un tiempo considerable que permite no sólo visualizar la escena en 3D (tarea principalmente a cargo de la GPU), si no que permite el uso de geometría mucho más complexa que la utilizada en esta prueba de concepto, la incorporación de sofisticados efectos de iluminación, y prácticamente cualquier efecto gráfico de un juego actual. Todo esto puede incluirse sin que ello represente ningún problema para la jugabilidad, controlada por el código original.

Respecto a los aspectos metodológicos que permitirían replicar este trabajo para cualquier otro juego, están limitados por el proceso de ingeniería inversa, que requiere la localización de la memoria de trabajo y la decodificación de la información almacenada en ella. La automatización de dicho trabajo sería de extrema complejidad, i evidentemente debe incluirse en el conjunto de tareas a considerarse en el trabajo futuro.

6. Conclusiones y Trabajo Futuro

Hemos conseguido desarrollar un remake de un juego clásico del ZX Spectrum sin reconstruir el juego completamente, sino añadiendo funcionalidades para generar un nuevo contexto gráfico, simplemente desviando el flujo gráfico e implementándolo nuevamente en un contexto 3D. Para ello hemos estudiado la arquitectura del ZX Spectrum, hecho uso de la in-

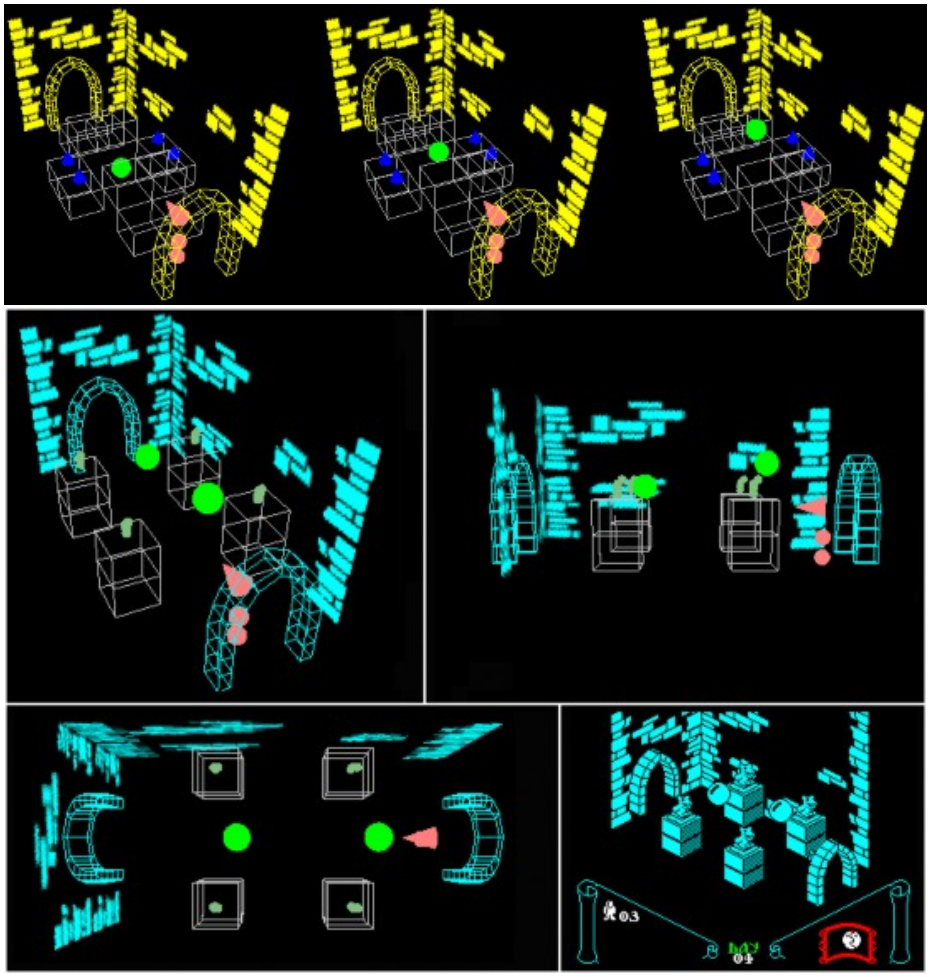

Figura 10. Abajo: Diferentes vistas de una habitación de Knight Lore 20XX y el original. Arriba: Secuencia de un balón en movimiento en una habitación.

geniería inversa para descubrir como trabaja el juego, y creado un nuevo motor de renderización basado en elementos geométricos y texturas.

Respecto a los trabajos futuros a desarrollar, además de los ya mencionados, éstos podrían ser dibujar los objetos mediante diferentes técnicas apropiadas para representarlos y conseguir un mejor acabado final de los mismos; aplicar fuentes de luz para mejorar el aspecto en consonancia a la temática del juego; mejorar la técnica de captura de sprites para que se obtengan directamente del código de juego; y la generalización del códi-

go para que se adapte a otros juegos que trabajen mediante las mismas técnicas.

Agradecimientos

Este trabajo ha sido parcialmente financiado con el proyecto TIN2014-52211-C2-2-R del Ministerio de Economía y Competitividad.

Referencias

1. Kopf, J., Lischinski, D.: Depixelizing pixel art. ACM Trans. Graph. 30(4), 99:1–99:8 (Jul 2011), http://doi.acm.org/10.1145/2010324.1964994
2. Lazo, J.M.: Iniciación al sistema filmation. MicroHobby 3(97, 98, 99 y 100), 28–30 (Octubre 1986), https://archive.org/details/microhobby-magazine
3. Racketboy: Enhance n64 graphics with emulation plugins texture packs. http://www.racketboy.com/retro/nintendo/n64/enhance-n64-graphics-with-emulation-plugins-texture-packs, accessed: 2016-05-26
4. Thibeault, C., Hervé, J.Y.: Object Detection in Emulated Console Games. In: Prakash, E. (ed.) 8th Annual International Conference on Computer Games, Multimedia and Allied Technology (CGAT 2015). Global Science & Technology Forum (GSTF), Global Science & Technology Forum (GSTF) (Apr 2015), https://www.dropbox.com/s/d25pkg6jai7cvn1/CGAT_2015_Proceedings_Paper_8.pdf?dl=0
5. Wild, C.: Knight lore data format. http://www.icemark.com/dataformats/knightlore/index.html, accessed: 2016-05-26