

Realidad Aumentada a partir de detección de rostros

Bernardi Martín Gustavo*, Remedi Augusto†, Rittano Ignacio‡

Universidad Nacional de Río Cuarto, Facultad de Ingeniería, Departamento de Telecomunicaciones

Río Cuarto, Córdoba, Argentina

*martin@mbernardi.com.ar

†agustoremedi@gmail.com

‡rittanoignacio@gmail.com

Resumen—En este trabajo se realizó una experiencia de realidad aumentada, consiste de una aplicación que a partir de la detección de cara de una persona, muestra imágenes en un monitor que producen una experiencia inmersiva. Mediante el uso de una cámara web fija, se toman imágenes que serán utilizadas para realizar la detección de la posición tridimensional de la cara del usuario en tiempo real. Es posible generar imágenes dinámicamente dependiendo de la posición detectada y generar una aproximación a una realidad aumentada, en la que las imágenes mostradas generan la sensación de que la pantalla es una ventana hacia un espacio tridimensional detrás de ésta.

I. INTRODUCCIÓN

El objetivo de este proyecto fue lograr la sensación de profundidad en una pantalla tradicional por medio de la visualización de imágenes que dependen de la posición de la cabeza del usuario. El primer punto principal del trabajo fue la detección precisa de la posición de la cara de la persona utilizando simplemente una cámara web y las técnicas estándar de detección. Además de la optimización de ésta y su posterior filtrado para lograr una detección en tiempo real y una experiencia suave.

El segundo punto importante fue la generación de imágenes a partir de esta estimación para lograr una sensación convincente de profundidad. La primera aproximación fue el mostrado de imágenes bidimensionales utilizando la técnica de paralaje, pero luego a partir de la utilización de librerías gráficas tridimensionales y del trabajo realizado por [1] se logró crear una experiencia más inmersiva en tres dimensiones.

Una aplicación posible de este proyecto es su colocación en una exposición de museo ya que es posible mostrar imágenes temáticas y educativas al mismo tiempo que se logra una experiencia corta e interesante, además no es necesario ningún elemento ni la interacción con elementos físicos que puedan deteriorarse. Como desventaja el efecto solo funciona para sólo una persona a la vez, pero de todos modos las personas que se encuentren cerca pueden también observar la información mostrada en pantalla.

El código fuente del proyecto se encuentra disponible online en <https://github.com/IgnaciooR/parallax-portal>.

II. DETECCIÓN DE ROSTROS

Para detectar la ubicación de la cara del usuario utilizamos el algoritmo Haar cascade implementado por OpenCV. Este algoritmo puede ser entrenado para la detección de cualquier

tipo de objetos, y para nuestro caso en particular, utilizamos un clasificador entrenado para la detección de caras.

El algoritmo va tomando secciones rectangulares o cuadradas de la imagen en forma secuencial y determinando si esa sección de la imagen es o no la cara de una persona. Comienza recorriendo la imagen secuencialmente analizando de a un cuadrado pequeño a la vez, y en cada pasada va analizando secciones más grandes de la imagen.

Si los cuadrados mínimos fuesen más pequeños se podrán encontrar caras más pequeñas en la imagen pero al mismo tiempo se afectará negativamente la performance. En este caso este tamaño mínimo es un indicador de cuán lejos se puede ubicar una persona.

Debido a que se va haciendo un barrido a lo largo de toda la imagen, es posible que alrededor de una misma cara encuentre múltiples positivos. De todos los resultados obtenidos, selecciona aquella que tenga la mayor intersección con el resto. Esto también es útil para evitar falsos positivos, haciendo que el algoritmo tenga que encontrar al menos una cantidad determinada de vecinos alrededor para poder afirmar que es una cara.

Todo este procedimiento se realiza de manera independiente para cada cuadro de video, por lo que generalmente se necesita ejecutar el algoritmo treinta veces por segundo y se necesita mucho procesamiento. Para reducir este costo computacional, ideamos unas optimizaciones que nos permitieron lograr la detección en tiempo real. Si en el cuadro de video anterior se detectó una cara, el próximo cuadro no será analizado de forma completa, se realiza un recorte de la imagen en la ubicación de la cara anterior con un 20 % de margen. De esta forma una vez localizada la cara de la persona, las siguientes búsquedas se harán solo en las proximidades de la última posición conocida. Otra optimización menor es que además de buscar en las cercanías de la detección anterior, sólo se buscarán caras con tamaños similares.

III. FILTRADO CON KALMAN

El filtro de Kalman es un filtro digital recursivo que se puede utilizar en sistemas dinámicos donde hay un cierto grado de incertidumbre acerca de lo que puede ocurrir en lo inmediato, debido a su poco requerimiento computacional es muy apto para problemas de detección en tiempo real. Además de permitir filtrar las detecciones y reducir su ruido, es capaz de realizar predicciones.

En nuestro caso el filtro de Kalman basado en [2], trabaja con las variables que se corresponden con la posición de la cara en el espacio tridimensional (X, Y, Z) y también con las velocidades en los respectivos ejes. A partir de conocer las detecciones pasadas, el filtro es capaz de predecir el siguiente estado y al mismo tiempo disminuir el ruido producido en la detección de cara.

La predicción de la posición a partir de las velocidades de movimiento conocidas no sólo es aprovechada para lograr una sensación de fluidez cuando se pierde la detección de la cara por momentos, sino también para interpolar muestras cuando la tasa de refresco del monitor es mayor que la tasa de refresco de la cámara. Otra utilidad del filtro es que al estimar la velocidad de la cabeza, tomamos del vector de estado la velocidad vertical para determinar si la persona ha realizado un salto.

IV. ARQUITECTURA DEL CÓDIGO

Este proyecto fue desarrollado en el lenguaje de programación Python, el código está compuesto por diversos archivos que fueron desarrollados de manera independiente para cumplir con distintos fines. Cada archivo además de contener funciones indispensables para el funcionamiento completo de la experiencia, contiene una pequeña demostración para comprobar el correcto funcionamiento de cada parte del programa.

Una característica importante del programa es que se utilizaron dos hilos o threads para separar en cuestiones de procesamiento dos partes importantes: la detección de caras y el dibujado de las imágenes. La finalidad detrás de la utilización de dos hilos yace en que los procesos de detección y dibujado se encuentran limitados en sus tasas de refresco. El hilo de la detección de caras va a estar limitado debido a los cuadros por segundo que soporta la cámara que se esté utilizando, mientras que para el dibujado el limitante está dado por la velocidad en el cual la pantalla puede mostrar las imágenes.

Por lo tanto al utilizar dos threads es posible desacoplar estos dos bucles para permitirles funcionar a su tasa máxima de refresco. Como se muestra en la Figura 1, el thread de detección almacena la posición de la última cara detectada mientras que el thread principal leerá esa posición cuando lo necesite, si la detección es más rápida que el dibujado habrán estimaciones de la posición de la cabeza que no se utilizarán para el filtrado o visualización. El caso más común es que la tasa de refresco de la pantalla sea mayor que la tasa de detección, por lo tanto si el thread principal determina que no hay una nueva ubicación de cabeza disponible, simplemente estimará la nueva posición realizando una predicción con el filtro de Kalman.

El programa necesita una gran cantidad de constantes y parámetros que deben ser ajustados manualmente para la puesta a punto y calibración, ya sea para corregir la detección de caras, filtro con Kalman, o dimensiones físicas de la pantalla. Encontramos muy útil crear de forma temprana una estructura que almacene todos estos parámetros y que permita su modificación de manera sencilla durante la ejecución del programa.

V. VISUALIZACIÓN

Se comenzó realizando una escena en dos dimensiones en forma de prototipo y a forma de realizar pruebas, pero es en la escena de tres dimensiones donde realmente se termina de apreciar la sensación de realidad aumentada.

V-A. Visualización bidimensional

Se creó una escena bidimensional en la cual se utilizó la técnica de paralaje para darle a las imágenes una sensación de profundidad. Paralaje se refiere a la diferencia en la posición aparente de un objeto según el ángulo en el que se lo mire.

Se utilizó la librería de Pygame para crear una ventana para la aplicación y dibujar las imágenes en ella. El efecto de paralaje y sensación de profundidad fue simulado mediante la modificación de la posición de las imágenes colocando a cada una de ellas un factor de movimiento, el cual modifica la sensibilidad de estas respecto a la posición de la cabeza. Como consecuencia, las imágenes que simulan ser cercanas tienen un bajo factor de movimiento, mientras las que están lejos, como el fondo, tienen un factor mayor.

Un problema de este método es que no permite realizar escenas con objetos tridimensionales complejos, además, la utilización de Pygame como librería gráfica no permite la aceleración por hardware y esto afecta en gran medida la fluidez de la aplicación en sistemas con limitada capacidad de procesamiento.

En la Figura 2 se muestra la escena creada a modo de ejemplo, consiste de un fondo a una gran profundidad y de figuras de personas a diversas profundidades. La temática es enumerar las personas que participaron en la Primera Junta de 1810 ubicados en el patio de la Casa de Tucumán, al mover la cabeza es posible seleccionar un prócer para visualizar más información sobre éste.

V-B. Visualización tridimensional

Como librería de dibujado utilizamos OpenGL y GLM, el cual permite el renderizado de escenas tridimensionales utilizando aceleración por hardware de video. Buscamos representar el interior de una casa colonial, al mismo tiempo que se muestran cartas con diversas estampillas o sellos postales históricos. En la Figura 3 se pueden observar capturas de pantalla.

OpenGL es principalmente utilizado para el desarrollo de videojuegos. En la gran mayoría de los casos se utiliza una visualización de perspectiva con la función `glm::perspective()`, pero en nuestro caso es necesario utilizar una matriz de proyección creada con `glm::frustum()`. La diferencia es que la visualización en perspectiva es simétrica hacia arriba y hacia los lados y es creada a partir de un cierto ángulo de campo de visión, mientras que en esta experiencia necesitamos una matriz de proyección asimétrica con ciertos parámetros. Esto se debe a que al jugar videojuegos el usuario siempre se ubica de frente a la pantalla, pero en nuestro trabajo el usuario se mueve y observa al monitor desde diferentes ángulos. Esto produce que la imagen se deforme y aparenta que está ubicada en el plano de la pantalla. Al usar una

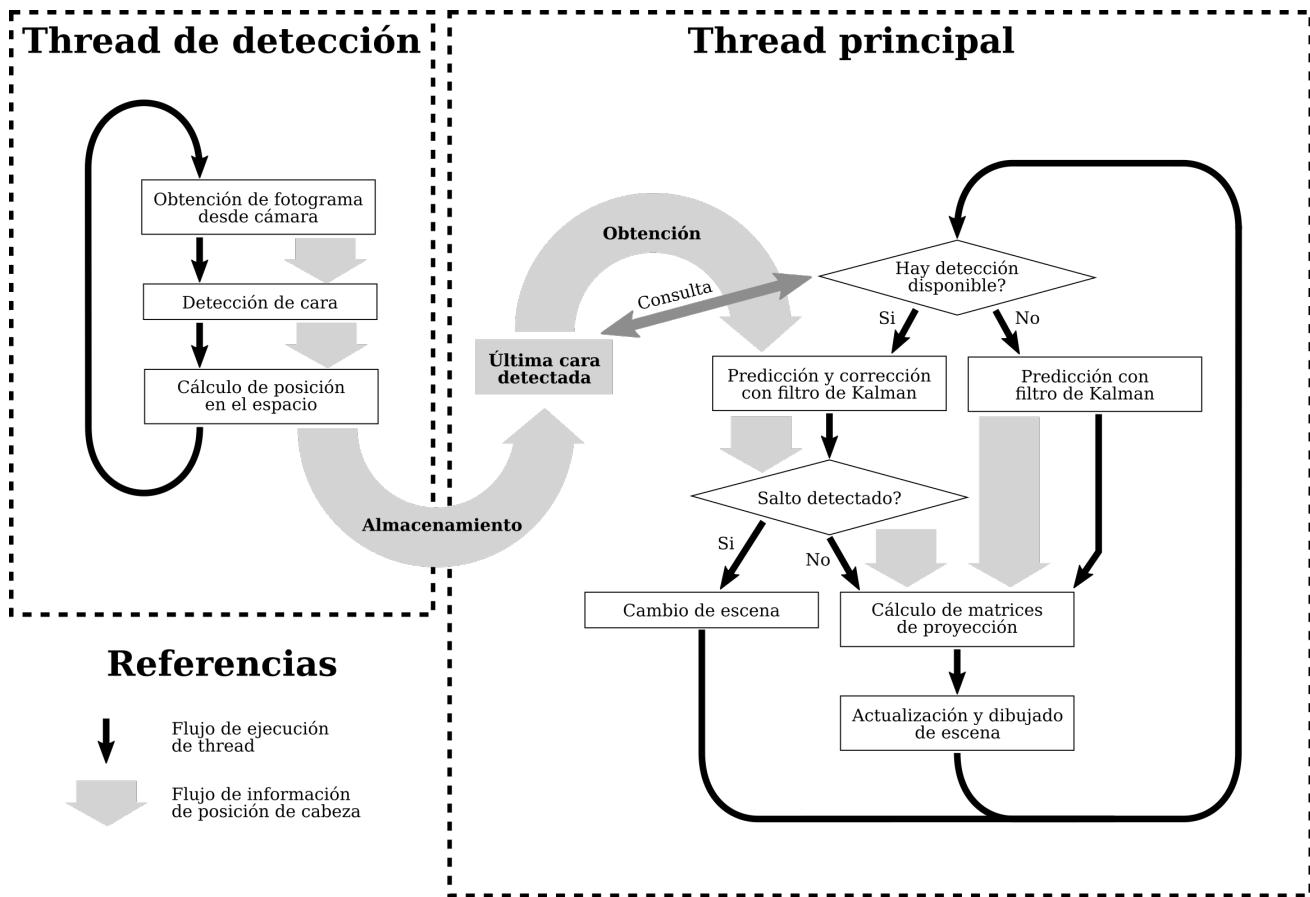


Figura 1. Flujo de ejecución del programa

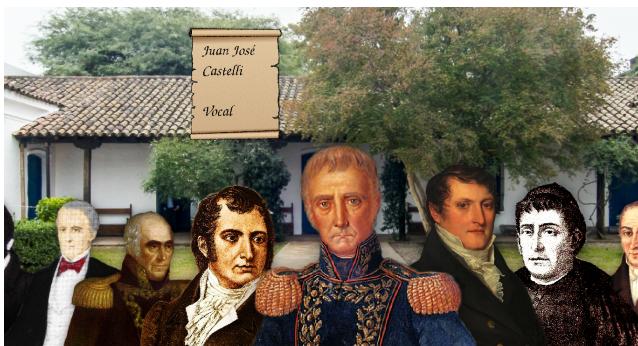


Figura 2. Visualización creada con Pygame para demostrar el efecto de paralelo. Imagen de fondo basada en foto de «Secretaría de Cultura de la Nación» [CC BY-SA 2.0].



Figura 3. Ejemplo de visualización de la escena 3D. Texturas basadas en imágenes de Sailko [CC BY-SA 4.0], «Darkwood67» (<https://www.deviantart.com/darkwood67/gallery/11280947>) y Øyvind Holmstad [CC BY-SA 4.0]

proyección asimétrica es posible compensar esta deformación de la imagen. La diferencia entre estos efectos se muestra en la Figura 4. Nuestra implementación es una traducción a Python del trabajo de [3], el cuál está basado en [1].

Respecto a la programación de las animaciones de la escena, las estampillas o sellos postales mostrados vuelan por la habitación siguiendo una trayectoria descripta por una curva paramétrica mostrada en la Figura 5, cuyas ecuaciones son:

$$\begin{cases} X(t) = t^5 \\ Y(t) = 0,2 * \cos(\pi * t/2) \\ Z(t) = 0,4 * \cos(\pi * t/2) \end{cases}$$

Para lograr un efecto de realidad aumentada convincente es útil colocar una caja impresa para lograr una sensación de continuidad hacia fuera de la pantalla del espacio tridimensional virtual, es importante al realizar el diseño la utilización de líneas rectas para demostrar de forma clara que la perspectiva

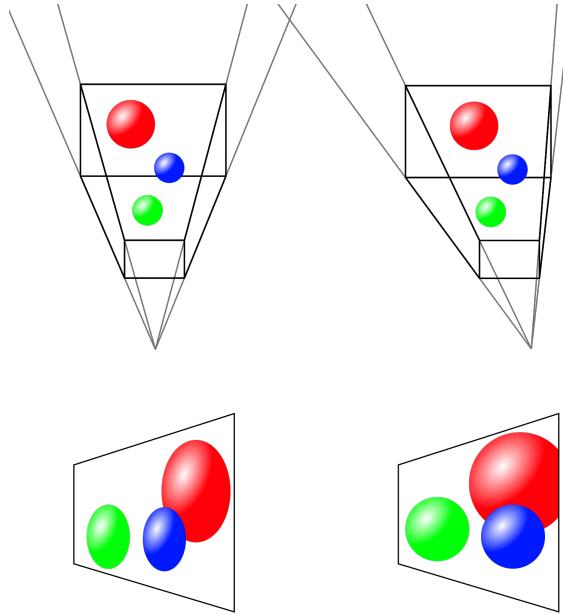


Figura 4. Visualización de las diferencias entre proyección simétrica con `glm::perspective()` (izquierda) y proyección asimétrica que compensa el ángulo de visión con `glm::frustum()` (derecha).

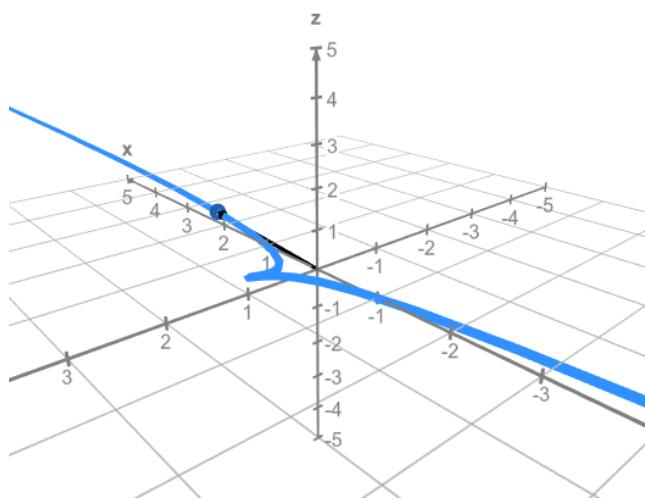


Figura 5. Visualización con [4] de curva paramétrica de la trayectoria de las estampillas en la habitación

se mantiene fuera y dentro de la pantalla. Nuestro prototipo se muestra en la Imagen 6, observar que de manera intencional las paredes contienen líneas rectas y que el piso es de baldosas cuadradas.

VI. CONCLUSIONES

Observamos que es posible realizar una experiencia de realidad aumentada de manera simple y económica por medio de la detección de la ubicación de la cabeza del usuario, la implementación se puede realizar en computadoras de gama baja con cámaras web de mediana definición. Es posible aplicar este método de manera directa en dispositivos con



Figura 6. Maqueta usada fuera de la pantalla para generar una continuidad del efecto de realidad aumentada. Observar que se está utilizando la cámara de una laptop para realizar la detección de la posición de la cabeza del usuario.

cámaras frontales como notebooks o teléfonos celulares, pero consideramos de interés su colocación en una galería de museo.

Como mejoras recomendamos descartar las librerías de dibujado bidimensional como Pygame y profundizar el trabajo en la generación de escenas tridimensionales con librerías ampliamente utilizadas para el desarrollo de videojuegos.

Consideramos que como proyecto futuro, la utilización de un televisor 3D no requerirá mucho trabajo de programación y producirá una gran mejora de la experiencia. De lo contrario, otra forma de lograr una sensación de profundidad convincente es la colocación de una caja alrededor de la pantalla para lograr una continuidad entre el espacio virtual y el real.

REFERENCIAS

- [1] R. Kooima, “Generalized Perspective Projection”. <https://csc.lsu.edu/~kooima/pdfs/gen-perspective.pdf>
- [2] R. Teammco, “Kalman Filter Simulation”. https://www.cs.utexas.edu/~teammco/misc/kalman_filter/
- [3] A. Girault, “ScreenReality”. <https://github.com/agirault/screenReality>
- [4] C. Chudzicki, “3D Parametric Curve Grapher”. https://christopherchudzicki.github.io/MathBox-Demos/parametric_curves_3D.html