Introducción al Procesamiento Digital de Imágenes

Universidad Nacional del Sur

Desarrollo de una aplicación de Realidad Aumentada

Martin Falco y Juan I. Pisula

Introducción

El presente trabajo surgió como una implementación básica del proyecto presentado durante la cursada de la materia a través de un canvas económico, el cual consiste en una galería de realidad aumentada donde se puede exponer arte digital.

En esta galería, no se verían lienzos con dibujos o pinturas, ni fotografías, ni grabados, si no una serie de imágenes denominadas glifos o marcadores. De esta manera, el público visitante recorrería la galería con la cámara de su smartphone apuntando a los distintos marcadores y estos serían reemplazados en la pantalla de su teléfono por una pieza de arte digital.

En el informe se plantea un algoritmo para detectar glifos en fotografías y reemplazar los píxeles del mismo por los píxeles correspondientes de una imagen sustituto así como su implementación en una aplicación de Android.

Finalmente se propone una mejora para la experiencia de realidad aumentada empleando *Image Based Rendering*, o Rendering Basado en Imágenes, lo que permitiría reemplazar los glifos no sólo por imágenes bidimensionales si no además por modelos en tres dimensiones.

Desarrollo

El primer paso fue investigar qué tipo de imagen es mejor utilizar como marcador, de manera de hacer su detección en la escena sencillamente. Se decidió utilizar marcadores cuadrados de 5x5 bloques, compuestos por una frontera y una imagen interior (de dimensiones 3x3) que representa un código binario. La información de este código binario es lo que permite decidir cuál va a ser la imagen a sustituir por el glifo en la fotografía tomada con un smartphone.

En estos glifos, los bloques que componen la frontera son de color negro mientras que los que componen la imagen interior pueden ser blancos o negros.

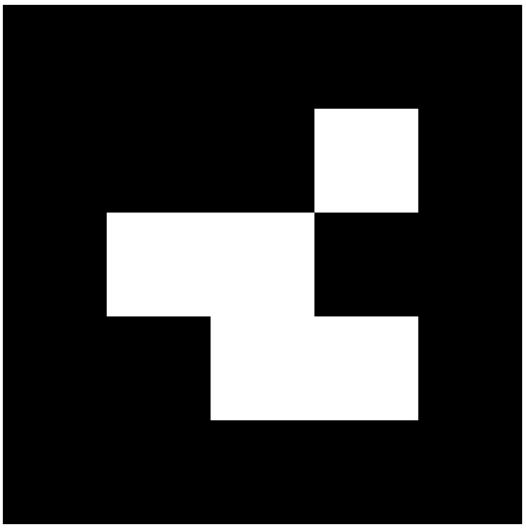

Figura 1: Ejemplo de glifo utilizado.

Como se mencionó anteriormente, la escena captada por un smartphone en esta situación puede contener glifos así como otros objetos que no son de interés detectar.

Aprovechando la forma de los marcadores elegidos, se procede a detectar si en la imagen de la escena existen contornos de cuatro lados. Esto puede subdividirse en 3 pasos:

1. Detección de bordes en la imagen.
2. Determinar los contornos conformados por los bordes.
3. Determinar cuáles de estos contornos pueden ser aproximados por polígonos de 4 lados.

Una vez que se obtienen los píxeles de la imagen que conforman a los distintos cuadriláteros hallados en la escena se debe verificar que pertenezcan a la frontera de alguno de los glifos y, si es así, se debe obtener el código binario que éste representa.

Antes de este paso es necesario realizar un pre procesamiento sobre los cuadriláteros. Lo más probable es que en la imagen tomada de la escena los glifos se encuentren rotados y con una cierta perspectiva. Sin embargo, para

la lectura del código binario, lo más sencillo es procesar una imagen de una vista frontal del marcador.

Teniendo las posiciones de los vértices de los polígonos, se determina una transformación geométrica denominada *homografía*, que mapea estos cuatro puntos a otros cuatro puntos que forman un cuadrado, tal como se muestra en la figura 2.

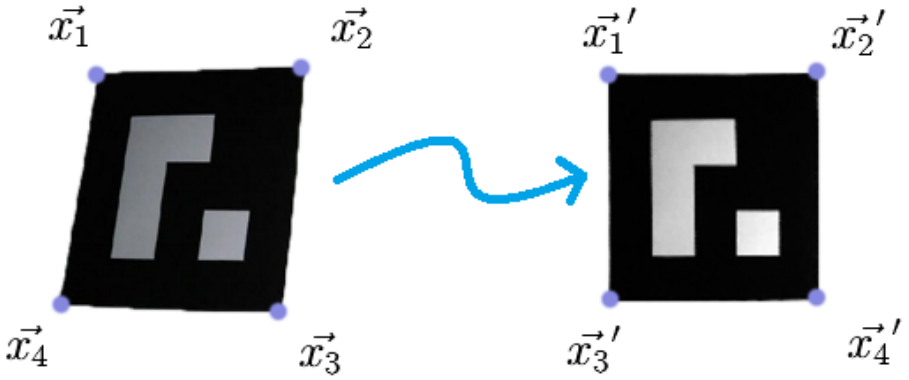

Figura 2: Transformación homográfica o de perspectiva del marcador detectado.

Utilizando la transformación encontrada, se le es aplicada a los cuadriláteros para obtener una vista frontal de los mismos.

Cabe destacar que como un contorno puede ser recorrido en dos sentidos, los marcadores que se detectan duplicados. Previo al procesamiento de los códigos binarios de los marcadores, sólo se seleccionan aquellos contornos de cuadriláteros cuyos vértices están ordenados en sentido horario.

A continuación se procesan estas nuevas vistas obtenidas. Primero se debe verificar que exista una frontera de bloques negros, y si es así, recorriendo los píxeles de la imagen se halla el código binario que representa.

El próximo paso consiste en determinar qué imagen sustituto se corresponde al código encontrado y hacer el reemplazo de píxeles. Para hacer esto, es necesario redimensionar la imagen sustituto de manera correcta y aplicar una transformación geométrica que mapea los puntos equivalentes a sus vértices, a los vértices del glifo correspondiente de la escena.

Implementación

El primer prototipo de la aplicación fue realizado en Python en un formato de script, para poder probar los pasos del procesamiento de una manera rápida y sencilla.

Luego, se portó el algoritmo desarrollado a una aplicación de Android. El desarrollo se hizo sobre la plataforma Android Studio. Si bien las aplicaciones en Android Studio se programan en Java, Android posee un SDK para utilizar código en C++, por lo que se decidió utilizar este último.

Tanto como en Python o C++ el procesamiento de imágenes se basó en la utilización de la librería *OpenCV*, que apunta a procesamiento en tiempo real y es de licencia gratuita.

La siguiente es una descripción genérica de la secuencia de sentencias del algoritmo desarrollado.

-Se convierte la imagen inicial a escala de grises, para eliminar información del color.
-Se aplica blur gaussiano para eliminar la información de alta frecuencia.
-Se efectúa detección de bordes con el algoritmo de Canny.
-De los bordes detectados, se extraen los trazos de los contornos. Nótese que aquí entra una imagen, pero la salida son arreglos de puntos.
-Para cada contorno detectado se ejecuta el siguiente código :
  -Se ordenan los vértices (calculando la mayor y la menor suma y resta de las coordenadas X e Y de cada punto).
  -Se calcula la homografía entre la imagen original y un cuadrado de lado predefinido.
  -Se efectúa la transformación de perspectiva de la imagen.
  -Se seleccionan los cuadriláteros cuyos vértices están ordenados en sentido horario.
  -Se verifica que el glifo sea válido, observando si posee una frontera de celdas negras.
  -Se obtiene el patrón de la imagen (se recorren los píxeles de cada celda del marcador, contando cuántos superan un valor de luminancia blanco y cuántos están por debajo de un umbral de luminancia de negro).
  -Se compara el patrón encontrado con los glifos predefinidos.
  -Si el glifo es valido y coincide con algún patrón predefinido, se ejecuta el siguiente código:
    -Se hace un resize de la imagen a superponer al alto y ancho máximo de la imagen del glifo detectado.
    -Se calcula la matriz de transformación necesaria para poner la imagen en perspectiva.
    -Se efectúa la transformación de perspectiva.
    -Se reemplazan los píxeles del glifo en la imagen de la escena por los píxeles de la imagen sustituto.

El desarrollo para Android se hizo sobre la plataforma de Android Studio. Si bien las aplicaciones en Android Studio se programan en Java, Android posee un SDK para utilizar codigo en C++, por lo que se decidió utilizar C++.

Particularmente para la implementación en C++, para la manipulación de datos se propuso trabajar parte del código utilizando tipos de datos pro-

pios de C++ (por ejemplo, un arreglo de arreglos de enteros), permitiendo un manejo más transparente.

Image Based Rendering

Se conoce como *ImageBasedRendering* a la serie de técnicas para capturar, representar y renderizar una escena a partir de imágenes de la misma. Este enfoque de síntesis de imágenes es una alternativa a las técnicas tradicionales basadas en geometría para lograr rendering fotorrealístico. En lugar de utilizar modelos geométricos 3D, se utilizan imágenes para renderizar las distintas posibles vistas de una escena.

Desde que existe IBR como área de estudio se han desarrollado diversas técnicas basadas en este concepto. Además, las estas técnicas suelen requerir más información que sólo imágenes. Se suele hacer una clasificación en tres categorías:

- Rendering sin geometría: El rendering de la escena se realiza a partir de imágenes sin ningún tipo de información geométrica. Estas técnicas se basan en la creación de nuevas vistas para la escena a partir de interpolación de imágenes previamente adquiridas, así como la utilización de mosaicos, etc.
- Rendering con geometría implícita: Las vistas nuevas son generadas a partir de correspondencia de características entre imágenes.
- Rendering con geometría explícita. Estas técnicas se basan en modelos geométricos precisos y pocas imágenes, así como información de profundidad para puntos en la imagen.

A continuación se hará una descripción de un método de rendering basado en puntos y el procedimiento a seguir para el uso esta técnica en aplicaciones de realidad aumentada.

Nubes de puntos

Este tipo de rendering permite generar una imagen a partir de una nube de puntos 3D (*points cloud*), esto es, un conjunto de vértices en un sistema de coordenadas tridimensional. La nube de puntos puede ser generada utilizando reconstrucción stereo, telemetría activa, structure from motion, así como scanners 3D, por lo que representan la superficie de la escena a renderizar.

Supóngase que se quiere renderizar una nube de puntos sobre una imagen o video que está siendo captada por la cámara de un smartphone. Se considera

que el smartphone se encuentra en un punto $\vec{x_a}$ en un sistema de coordenadas de tres dimensiones. A su vez, la cámara del smartphone está apuntando a una dirección particular, captando la luz que incide sobre su sensor óptico para poder formar una imagen de la escena.

Se puede realizar la simplificación que toda la luz detectada por el sensor óptico proviene de un único plano, el plano de imagen, que se ubica a una determinada distancia del sensor. De este modo se tiene que los distintos elementos visibles en la imagen son la proyección de los elementos de la escena sobre el plano de imagen. Además, se considera que todos los píxeles de la imagen coinciden con puntos de este plano.

Este concepto es una aproximación básica de la *función plenóptica* del cono de rayos visible desde cualquier punto del espacio, en cualquier dirección, cualquier momento en el tiempo, para cualquier longitud de onda. Esta función plenóptica es una función $P = P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$, donde los ángulos θ, ϕ determinan la dirección en la que se observa; λ indica la longitud de onda del rayo de luz; t es el instante de tiempo; V_x, V_y, V_z indican las coordenadas del espacio donde se sitúa el observador, y describe toda la energía radiante que puede ser percibida desde el punto de vista del mismo.

Habiendo introducido esto, se puede considerar que un sensor fotográfico digital lo que hace es muestrear el valor de la función plenóptica en puntos (x, y) (en lugar de coordenadas (θ, ϕ)) del plano de imagen para las longitudes de onda correspondientes al color rojo, verde y azul. De esta manera, la función plenóptica que se describe es una función $P = P(x, y, \lambda)$ que posee tanto un dominio como rango discretos.

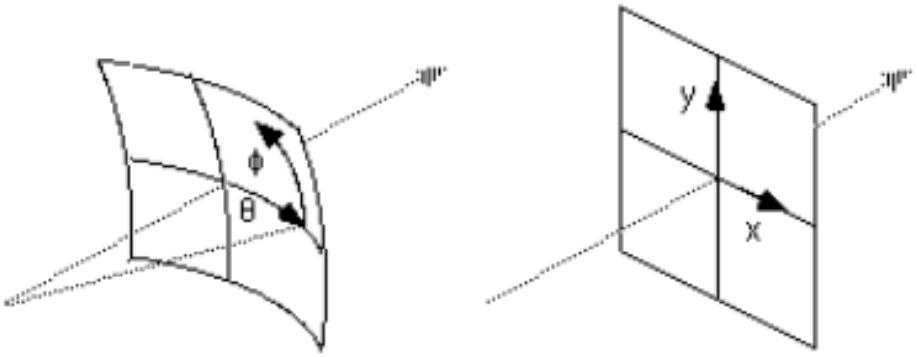

Figura 3: Parametrización angular y cartesiana de la función plenóptica. Si bien la parametrización cartesiana es más apta para aplicaciones de visión computacional, un enfoque angular es mejor para describir toda la esfera de información óptica en un punto del espacio.

Ahora, si se quisiera añadir a la imagen los distintos puntos $\vec{x_i}$ que conforman a la nube de puntos en cuestión, sólo se requeriría proyectar cada uno de estos puntos al plano de imagen.

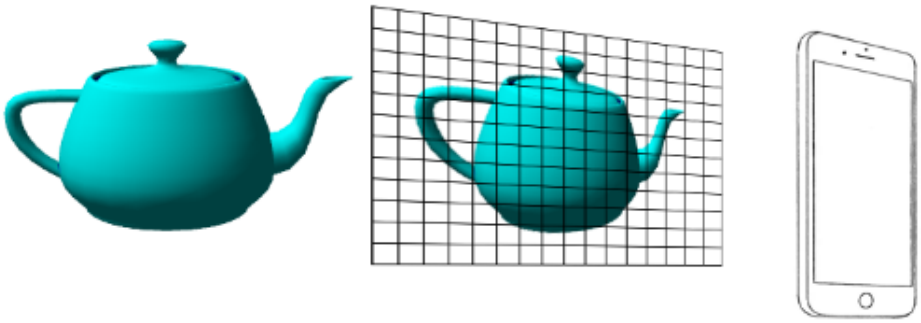

Figura 4: Proyección de un conjunto de puntos al plano de imagen.

Si para realizar esta serie de operaciones se considera, por ejemplo, que la dirección Z del sistema de coordenadas crece en la dirección que apunta la cámara del Smartphone y que a los puntos $\vec{x_i}$ se les realiza una proyección ortogonal sobre el plano de imagen, las nuevas coordenadas (x, y) del punto $\vec{x_i}'$ en el plano de imagen son las mismas que las de $\vec{x_i}$.

Para obtener vistas del objeto desde otras direcciones se proponen dos maneras: o bien cambiar el plano donde es proyectada la nube de puntos y determinar qué coordenadas de ese plano representan los píxeles de la imagen captada por la cámara; o rotar los puntos de la nube tal que al realizar la proyección sobre el plano de imagen se obtenga la vista deseada.

Cabe mencionar que esta técnica de rendering no es perfecta. Por lo general los puntos $\vec{x_i}'$ se encuentran en posiciones entre píxeles. La discretización de los valores (x, y) posibles puede llevar a que ocurran huecos en la imagen. Existen otras causas por las que pueden ocurrir huecos, como los son el escalado de la nube de puntos o discontinuidades en la adquisición de los datos.

Además del problema de huecos, es necesario lidiar con el problema de los puntos $\vec{x_i}$ que al ser proyectados al plano de imagen deben ocupar un mismo píxel en pantalla. La alternativa más sencilla es utilizar un Z-buffer para determinar el color del píxel en cuestión. De esta manera el píxel tomará el color del punto $\vec{x_i}$ más cercano al plano de imagen.

Hasta aquí no se ha mencionado cuál es la ubicación de los puntos $\vec{x_i}$ con respecto a la ubicación de la cámara $\vec{x_a}$. A priori, no resulta posible saber esto ya que los puntos de la nube pueden ser adquiridos a través de distintas técnicas e incluso para una misma técnica la adquisición puede realizarse de distintas maneras, teniendo diferentes orígenes de coordenadas. Además, el punto $\vec{x_i}$ se ha seleccionado de manera arbitraria por lo que esta ubicación no brinda información relevante.

Otro detalle omitido es la escala del objeto representado por la nube de puntos con respecto a los objetos en la imagen de la cámara. Realizar operaciones de escalado y traslación sobre los puntos $\vec{x_i}$ resultan necesarias para hacer el rendering acorde.

Implementación

En el repositorio de GitHub de este proyecto se encuentran las funciones implementadas en Python para hacer el rendering de una nube de puntos así como un script demostrativo (más información en la sección *Resultados*).

Para facilitar el manejo de datos se trabajó con un archivo *.txt* que posee información en formato ASCII de las coordenadas (x, y, z) de los puntos pertenecientes a una nube.

La siguiente es una descripción genérica de la secuencia de sentencias del algoritmo desarrollado para renderizar una nube de puntos en una imagen en escala de grises.

-Se rotan los puntos pertenecientes a la nube en la dirección deseada.
-Se interpolan linealmente los valores de todas las componentes z de los vértices, para que estos queden en el rango entre 0 y 255.
-Se crea una imagen cuyo ancho es la distancia entre los vértices más alejados según la componente x, y cuyo alto es la distancia entre los vértices más alejados según la componente y.
-Para cada vértice (x_i, y_i, z_i) de la nube se accede al píxel x_i, y_i de la imagen y se le asigna el valor z_i. Si existen vértices con las mismas coordenadas según x e y, al píxel de la imagen se le es asignado el máximo valor según z entre estos puntos.

Para reemplazar la imagen obtenida sobre un marcador, la imagen se escala y se posiciona centrada sobre el mismo.

Resultados

Además de la aplicación de Android, para ilustrar la estrategia propuesta para detectar y reemplazar marcadores por imágenes o por nubes de puntos se elaboró un *notebook* de IPython disponible en el repositorio de GitHub de este proyecto. El mismo posee el código explicado y comentado paso por paso, así como visualizaciones en las etapas intermedias del procesamiento.

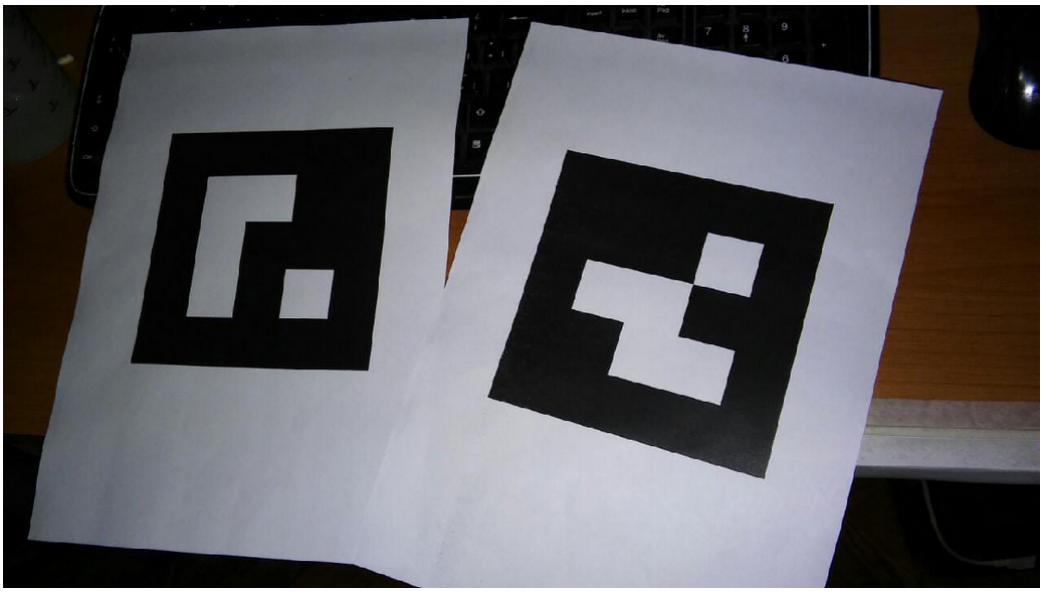

Figura 5: Imagen de la escena donde se sitúan los marcadores.

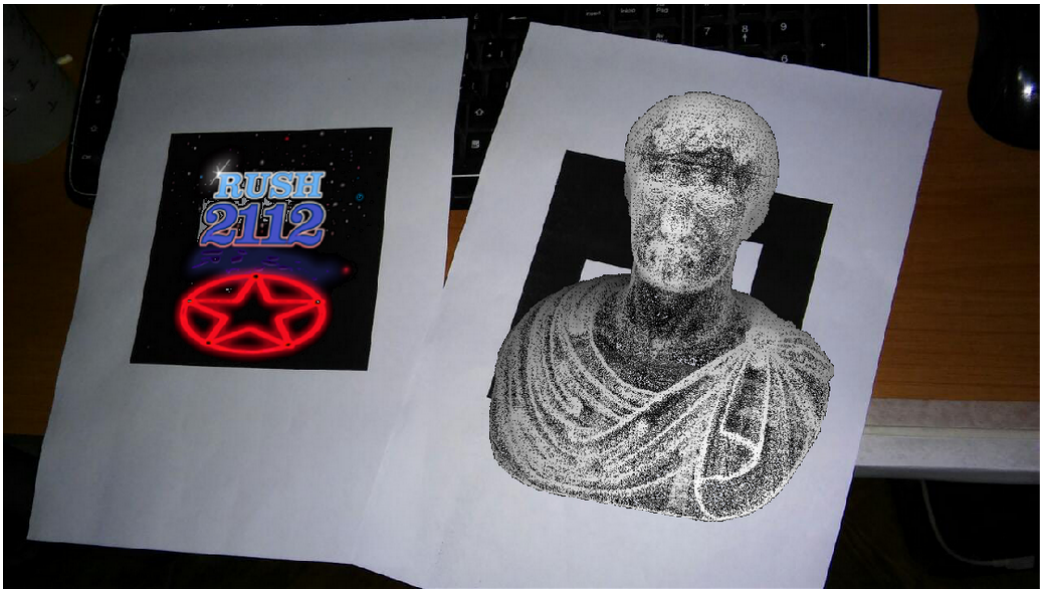

Figura 6: Escena con los marcadores ya reemplazados.

Conclusiones y trabajo futuro

Aunque este proyecto no abarca mucho procesamiento de imágenes, durante todo su desarrollo se aprendieron conceptos aplicados a visión computacional y computación gráfica.

Se propone analizar otras alternativas a los pasos seguidos en el algoritmo de reconocimiento de marcadores. Por ejemplo, en lugar de binarizar las imágenes de los glifos detectados utilizando valores de umbral arbitrarios, se podría estudiar el uso de otros métodos de binarización, como lo es el algoritmo de Otsu. Así mismo, puede que existan procedimientos más correctos para determinar el código binario a partir de la vista frontal de un marcador.

La utilización de otros algoritmos también implica estudiar qué tan complejos son para realizar este trabajo con un smartphone en tiempo real. Otro de los pasos a seguir es el estudio de cómo disminuir la latencia de la aplicación en dispositivos móviles.

Se propone también continuar la línea de trabajo en detección de marcadores así como su reemplazo cuando estos se encuentran ocluidos por otros objetos.

Para el manejo de modelos 3D y nubes de puntos se recomienda el uso de archivos en los formatos standard en lugar de *.txt*, como lo son *.ply*, *.pcd*, *.obj*. En este trabajo se implementaron en Python algoritmos de rotación y rendering para nubes de puntos, cuando existen herramientas que poseen

estas operaciones optimizadas, como lo es *OpenGL*.

Además, para mejorar la experiencia de realidad aumentada en aplicaciones con modelos 3D, es necesario hacer estimación de pose del marcador detectado para rotar el modelo adecuadamente. Para esto es necesario tener disponibles los parámetros de la matriz de la cámara que capta la escena. Estos parámetros son los que describen cómo una cámara mapea puntos 3D del mundo a puntos 2D en la imagen. En la proyección de la nube de puntos al plano de imagen se utilizó una proyección paralela, sin embargo tiene sentido analizar el uso de otras proyecciones.

Por último se menciona que el estado del arte en el desarrollo de aplicaciones móviles de realidad aumentada está en las librerías *ARToolKit* y *ARKit* publicadas por Google y Apple respectivamente. Si bien su compatibilidad con smartphones de generaciones anteriores es limitada, estas son las opciones que se deberían utilizar para hacer una aplicación comercial, en lugar de hacer un desarrollo total.

Referencias

[1] OpenCV documentation. Disponible en https://docs.opencv.org/3.3.0/index.html.

[2] Edward H. Adelson and James R. Bergen. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, pages 3–20. MIT Press, 1991.

[3] Sing Bing Kang Heung-Yeung Shum, Shing-Chow Chan. *Image-Based Rendering*. Springer, 2007.

[4] Ross D. Milligan. Electric Soup. Disponible en https://rdmilligan.wordpress.com.

[5] F.J Madrid-Cuevas M.J. Marín-Jiménez S. Garrido-Jurado, R. Muñoz-Salinas. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, jun 2014.

Anexo

Repositorio de GitHub del proyecto disponible en :
https://github.com/juanigp/Proyecto-final-IPDI