Realidad Aumentada
Programación OpenGL
Laberinto 3D

César Bragagnini
Email: cesarbrma91@gmail.com

I. Introducción

La realidad aumentada consiste en la combinación del entorno físico real con un entorno virtual, creando de esta manera una realidad mixta en tiempo real. Existen diversos equipos que pueden usarse para la generar realidad aumentada, como móviles, lentes, tablets, etc.; siendo el principal componente la cámara, que permite obtener información del entorno físico para ser procesada y generar los elementos virtuales correspondientes.

La finalidad de este trabajo es generar realidad aumentada utilizando el patrón de círculos de calibración de cámara y los parámetros intrínsecos y extrínsecos de la cámara para generar el espacio del mundo en OpenGL. Sobre el patrón se proyectarán objetos 3D que tienen propiedades gráficas como la transparencia y la iluminación; además que estos objetos deben seguir el movimiento del patrón para parecer lo más real posible.

El presente informe está organizado en cuatro secciones: la Sección II detalla algunos conceptos básicos que fueron necesarios para realizar la implementación de este proyecto; en la Sección III se detalla la implementación y consideraciones tomadas para generar el ambiente OpenGL, los objetos 3D y sus propiedades gráficas. En la Sección IV se muestran los resultados obtenidos hasta el momento y en la Sección V se encuentran las conclusiones a las que se ha llegado después de realizar este trabajo.

II. Conceptos relacionados

II-A. Matriz del modelo (M)

El modelo [3] está definido por el conjunto de vértices del objeto; las coordenadas X, Y y Z de los vértices se encuentran definidos en relación al centro del objeto, que generalmente sería el punto $(0, 0, 0)$ (Ver Figura 1).

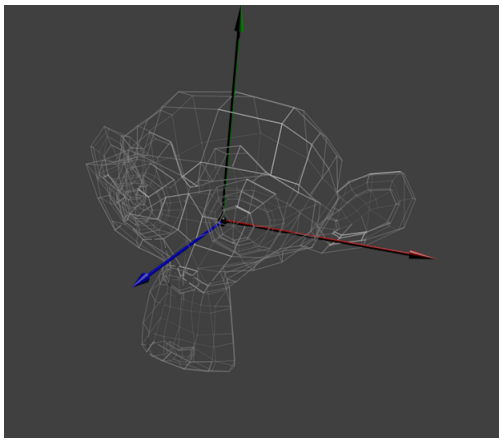


Figura 1. La matriz del modelo representa el conjunto de vérticees del objeto.

II-B. Matriz de la vista (V)

La matriz vista define la posición de la cámara en relación al mundo, si se desea mover la cámara se debe ingresar una nueva matriz que posicione la cámara. (Ver Figura 2)

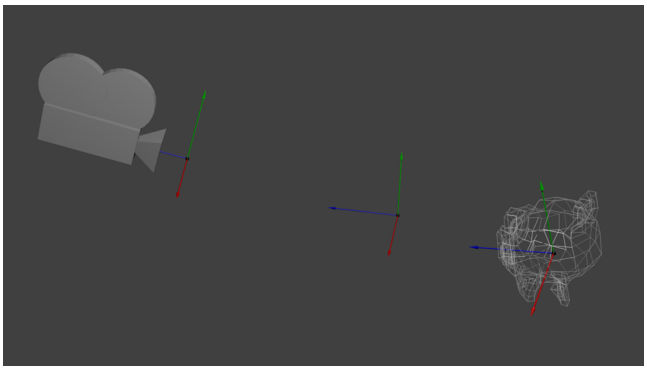


Figura 2. La matriz de la vista representa la posición de la cámara en el mundo.

II-C. Matriz de proyección (P)

Representa el espacio de la cámara [3], permite definir una área de escena que será visualizada a través de la cámara y se basa en la información de las coordenadas x,

y y la distancia a la cámara z para renderizar un objeto. (Ver Figura 3)

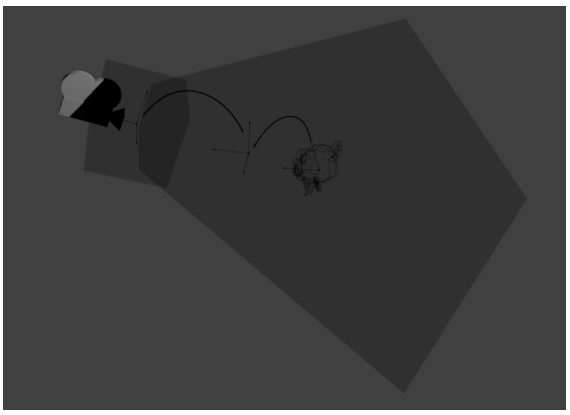


Figura 3. La matriz de proyección permite representar el mundo de la cámara.

III. Implementación

III-A. Uso de los parámetros intrínsecos

Para esta etapa se ha cargado el archivo XML que contiene los datos de los parámetros intrínsecos de la cámara, dimensiones del frame, tamaño del patrón (número de filas y columnas) y los coeficientes de distorsión. La matriz de parámetros intrínsecos K requiere algunas modificaciones previas antes de ser convertida en la matriz de proyección para OpenGL: la primera modificación consiste en cambiar la tercera columna a negativo porque la cámara de OpenGL mira hacia el eje z-negativo.

$$K = \begin{bmatrix} fx & sk & -cx \\ 0 & fy & -cy \\ 0 & 0 & -1 \end{bmatrix}$$

La segunda modificación consiste en agregar una columna y final adicional para evitar la pérdida de información de profundidad (eje Z).

$$Persp = \begin{bmatrix} fx & sk & -cx & 0 \\ 0 & fy & -cy & 0 \\ 0 & 0 & np+fp & np*fp \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Donde np es el valor más próximo a la cámara y fp es el valor máximo hasta donde la cámara logrará visualizar un objeto.

Finalmente, transformamos la matriz K de OpenCV a una matriz OpenGL, multiplicando la matriz ortogonal de OpenGL con la matriz $Perp$ resultante de la modificación de K [4]. La matriz de proyección resultante es:

$$Proj = \begin{bmatrix} \frac{2*fx}{w} & \frac{2*sk}{w} & 1-\frac{2*cx}{w} & 0 \\ 0 & \frac{2*fy}{h} & \frac{2*cy}{h}-1 & 0 \\ 0 & 0 & -\frac{(fp+np)}{fp-np} & -\frac{(2*fp*np)}{fp-np} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Debido a que OpenGL trabaja con un ordenamiento por *column-major*, aplicamos la transpuesta a la matriz para poder realizar las demás operaciones.

Adicionalmente, se debe definir la matriz de vista para ubicar la cámara en el mundo, en nuestro caso la hemos definido de la siguiente manera:

- Ubicación de la cámara: $(0, 0, 1)$ (eje Z).
- Punto de visión de la cámara: $(0, 0, 0,)$ (centro del mundo).
- La cámara visualiza al eje $y+$.

La definición de estas matrices se realiza una sola vez; en todo el proceso estas matrices son las mismas.

III-B. Uso de los parámetros extrínsecos

En el procesamiento de cada frame, por cada frame se aplica el siguiente procesamiento:

- Filtrado: Se convierte la imagen a escalas de grises, y se aplica un filtro Gaussiano, lo que permite la limpieza del ruido en la imagen, pero sin afectar mucho los contornos del patrón.
- Segmentacion: Se implemento la umbralizacion adaptativa propuesta por (x,y) [1].
- Obtención de los contornos, para el padron de anillos al tener 2 anillos concéntricos, por lo tanto existe una jerarquia de contornos(padre e hijo), adicionalmente el círculo hijo no tendrá hermanos en su mismo nivel y no tendrá hijos; de esta manera se puede eliminar contornos que no cumplan con esta característica y se reduciría el ruido.
- Reduccion de ruido, al obtener los centros de contorno en la etapa aun existe mucho ruido para esto se uso distintas heurísticas busqueda en profundida, arbol de recorrido mínimo, media aritmética, y por ultimo el filtro de Kalman como predictor del centroide del padrón.
- Etiquetado, que es ordenamiento de los key points; para esto se hizo lo siguiente: se procede a aplicar ConvexHull al conjunto de los puntos del patrón; se halla las esquinas del convexhull puesto que éste será esencial en la detección de las rectas de los extremos del patrón de anillos; una vez

que conseguimos obtener las esquinas del patrón mediante el convexhull, empezamos a hallar rectas que intersequen con 6 puntos colineales, para ello aprovechamos la secuencia de puntos ordenada que nos da el convexhull; tomamos dos puntos consecutivos del convexhull para crear una recta y detectar los puntos que intersecan a la recta, si son 6 los puntos, ésta será considerada como recta que esta en el extremo del patrón; una vez que tenemos las 2 rectas de 6 puntos de cada una, que conforman los extremos del patrón de anillos, ordenamos estos puntos de manera ascendente con respecto a x y de manera descendente con respecto a y, para después trazar rectas entre los elementos ordenados de cada arreglo; una vez trazadas las rectas de extremo a extremo, empezamos a hallar los puntos que intersecan dicha recta para completar los puntos del patrón por completo.

- Obtención del vector de rotación y traslación; para calcularlos se necesita la distribución del padrón en el mundo real, juntos con la matriz de cámara y los coeficientes de distorsión, se uso la funcion $solvePnP$ de OpenCV una vez obtenido el vector de rotación se uso la función $Rodriguez$ de OpenCV para convertir el vector de rotación en una matriz de 3x3

Una vez que se ha obtenido la matriz de rotación R y el vector de traslación t, se genera la matriz M de OpenGL (matriz del modelo) que nos permitirá rotar y trasladar un objeto 3D para seguir el movimiento del patrón. La matriz generada para este fin es la siguiente:

$$M = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ -r_{21} & -r_{22} & -r_{23} & -t_2 \\ -r_{31} & -r_{32} & -r_{33} & -t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Debido a que el movimiento del patrón varía en cada frame, matriz M es actualizada durante cada frame para que los objetos sea renderizados correctamente.

III-C. Transformaciones geométricas

Cuando un objeto es creado sin indicar una posición inicial, se ubica en el origen del mundo (punto $(0, 0, 0)$). (Ver Figura4).

Sin embargo, en este tipo de aplicaciones se busca la interacción del usuario de manera que pueda interactuar con los objetos virtuales, es decir, moverlos, escalarlos, rotarlos, etc. La ejecución de cualquiera de estas interacciones implica modificar el objeto 3D, para ello se requiere aplicar transformaciones geométricas en un orden correcto para que el comportamiento sea el esperado. En este caso OpenGL permite realizar multiplicaciones de matrices para realizar una transformación y el orden en que se ejecuta es de izquierda a derecha. Por ejemplo:

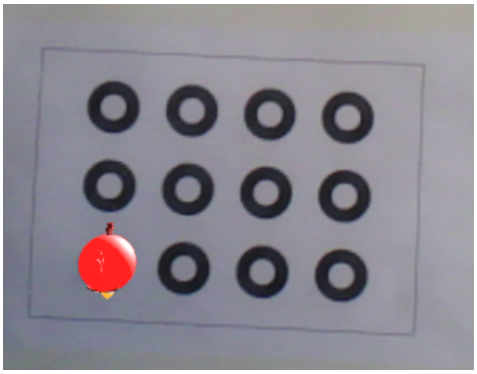


Figura 4. Objeto 3D creado sin indicar su posición inicial, aparecerá por defecto en el origen de coordenadas de este mundo.

$$MVP = P * V * M$$

para obtener la matriz final MVP que es la resultante de multiplicar la matriz de proyección, con la matriz de vista y finalmente con la matriz del modelo.

En esta implementación se permite la traslación de los objetos 3D en todo el mundo virtual, además de moverse junto con el patrón, puede posicionarse en algún otro lugar respetando ese comportamiento; para este caso este desplazamiento representado por un vector 3D (x, y, z) es convertido a una matriz 4×4 para poder operarse con las matrices M, V y P; finalmente el comportamiento del objeto es modelado con la nueva matriz MVP:

$$MVP = P*V*M*translate(mat4(), vec3(tx, ty, tz))$$

donde tx, ty y tz indica el desplazamiento en cada uno de los ejes del mundo.

III-D. Pipeline implementado

La Figura 7 muestra el pipeline usado.

III-E. Descripción del juego

Se simula con realidad aumentada el juego de la laberinto donde por inclinacion de la plataforma y la gravedad, hacen rodar la bolita por los distintos caminos; el objetivo del juego es hacer llegar la bolita a un agujero que es la salida del laberinto, ver la Figura 6 para mayor entendimiento.

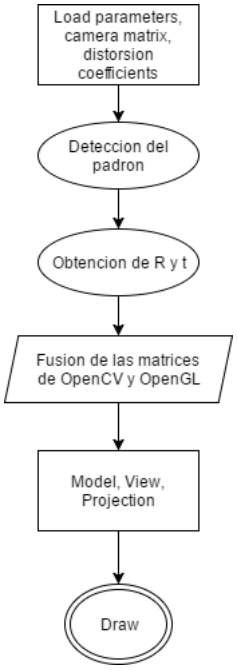


Figura 5. Pipeline OpenCV + OpenGL.

Figura 6. Juego de laberinto para niños

III-F. Angulos de Euler

Los ángulos de Euler constituyen un conjunto de tres coordenadas angulares que sirven para especificar la orientación de un sistema de referencia de ejes ortogonales, normalmente móvil, respecto a otro sistema de referencia de ejes ortogonales normalmente fijos. Estos angulos $(\theta_1, \theta_2, \theta_3)$ forman los angulos respecto a los ejes X,Y,Z. Se calculos estos angulos de acuerdo a la matriz de rotacion del padron. La Figura **??** muestra el codigo usado para obtener los angulos de Euler a partir de una matriz de rotación.

De acuerdo a la variacion de $theta_2$ se determino cuando una pelotita debia avanzar por +X o -X, y por la variacion de $theta_1$ determina cuando la pelotita avanza por +Y o -Y. El rango de toleracia: [0-0.15] grados para

```
glm::vec3 rotationMatrixToEulerAngles(Mat &R)
{
    float sy = sqrt(R.at<double>(0, 0) * R.at<double>(0,
      0) + R.at<double>(1, 0) * R.at<double>(1, 0));
    bool singular = sy < 1e-6;
    float x, y, z;
    if (!singular){
        x = atan2(R.at<double>(2, 1), R.at<double>(2, 2));
        y = atan2(-R.at<double>(2, 0), sy);
        z = atan2(R.at<double>(1, 0), R.at<double>(0, 0));
    }else{
        x = atan2(-R.at<double>(1, 2), R.at<double>(1,
          1));
        y = atan2(-R.at<double>(2, 0), sy);
        z = 0;
    }
    return glm::vec3(x, y, z);
}
```

Figura 7. Codigo para obtener los angulos de Euler a partir de una matriz de Rotación

$abs(\theta_1)$ significa que no hay variacion de la posicion de la pelotita respecto al eje Y, y el rango de tolerancia [3.0616-3.1416] grados para $abs(\theta_2)$ significa que no hay variacion de la posicion de la pelotita respecto al eje X.

III-G. Trabajos futuros

Implementar una variacion de la pelotita en sus 3 coordenadas usando la siguiente formula:

$$V_{bolita} = (V_x, V_y, |proj_{\mathbf{U}_{\theta_1}} \mathbf{g}| + |proj_{\mathbf{U}_{\theta_2}} \mathbf{g}|)$$

Donde V_x es la variacion constante en el eje x(puede ser negativa dependiendo de la inclinación), V_y es la variación constante en el eje y(puede ser negativa dependiendo de la inclinación), $|proj_{\mathbf{U}_{\theta_1}} \mathbf{g}|$ es la norma de la proyección de la g(gravedad) sobre el vector formado por el angulo $theta_1$ y el eje y, $|proj_{\mathbf{U}_{\theta_2}} \mathbf{g}|$ es la norma de la proyección de la g(gravedad) sobre el vector formado por el angulo $theta_2$ y el eje x; considerar que la gravedad es el eje -Z. Se piensa usar teoria de colisiones y reaccciones.

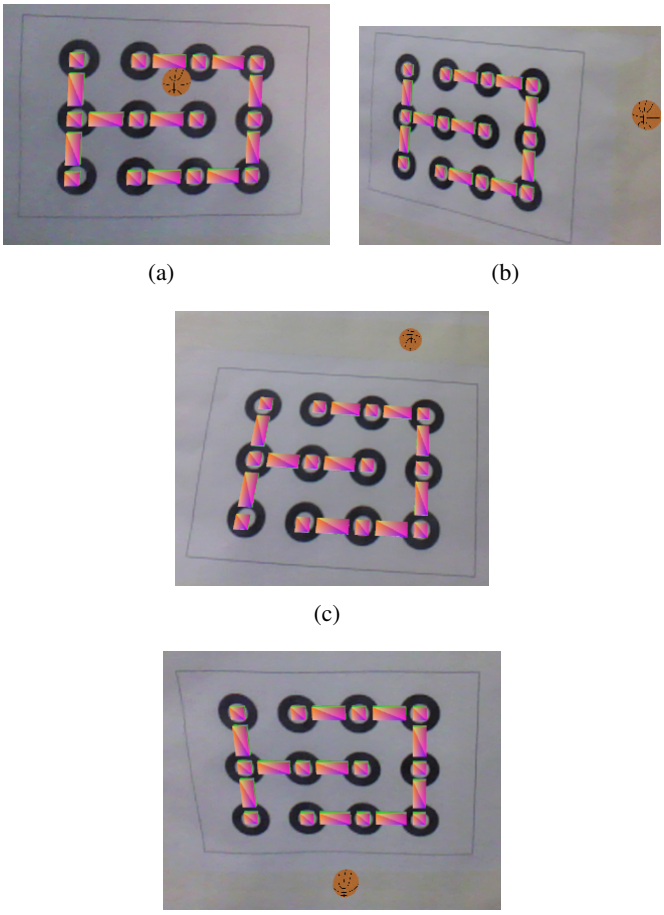
IV. Resultados

IV-A. Seguimiento del juego

En la Figura 8 se muestran algunas imagenes del juego.

V. Conclusiones

- Es importante entender los parámetros de cámara y coeficientes de distorsión para poder pasar los datos del mundo OpenCV al mundo OpenGL.
- La obtencion de inclinaciones es facil de conseguir, usando angulos de Euler; se puede usar cuartenios.
- El uso del tiempo real, esta sujeto al hardware empleado, pero tambien al programa implementado



(a) (b)

(c)

(d)

Figura 8. Se muestra 4 inclinaciones del padron que hacen variar el movimiento de la pelota en eje X y eje Y

donde el mayor tiempo de procesamiento esta en procesar la imagen, la parte grafica de OpenGL no es costosa.

REFERENCIAS

[1] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 3rd edition, 2007. ISBN 013168728X,9780131687288.

[2] M.-A. T. F. L. Guen. *Presentacion: Iluminacion*. 2017.

[3] opengl tutorial. Tutorial 3 : Matrices. http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/.

[4] K. Simek. Calibrated Cameras in OpenGL without glFrustum. http://ksimek.github.io/2013/06/03/calibrated_cameras_in_opengl/.