"Año de la unidad, la paz y el desarrollo".

UNSA

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

Escuela Profesional de Ciencia de la Computación

Computación Gráfica

Proyecto Final

Integrantes:
- Katherine Nikole Béjar Román
- Sennayda Rimache Choquehuanca
- Merisabel Ruelas Quenaya
- Anagrabriela Pilar Jiménez López

Docente:
Rosa Yuliana Gabriela Paccotacya Yanque

30 de julio de 2024

Índice

1. Introducción

El presente proyecto tiene como objetivo desarrollar un sistema de realidad aumentada utilizando las bibliotecas OpenCV y OpenGL en C++. La realidad aumentada es una tecnología que superpone objetos virtuales en el mundo real, proporcionando una interacción enriquecida entre el entorno físico y los elementos digitales. En este caso, se implementa un programa que detecta un patrón de tablero de ajedrez y superpone un objeto tridimensional sobre dicho patrón en tiempo real.

La realidad aumentada (AR) es una tecnología innovadora que mezcla el mundo digital con el entorno físico en tiempo real. Utilizando cámaras y pantallas, la AR permite superponer imágenes, videos y modelos 3D sobre lo que vemos en el mundo real, creando experiencias interactivas y envolventes. Este proyecto tiene como objetivo configurar y ejecutar una aplicación de realidad aumentada en Linux Mint, que detecta marcadores específicos y muestra objetos 3D sobre ellos.

La AR ha encontrado aplicaciones en diversos campos, desde el entretenimiento y la educación hasta la visualización de datos. En juegos móviles, manuales interactivos y herramientas de diseño, la AR ha demostrado su capacidad para mejorar la forma en que interactuamos con la tecnología.

El sistema de realidad aumentada desarrollado detecta un patrón específico utilizando la cámara web. Una vez detectado el patrón, el programa calcula la posición y orientación del patrón en el espacio tridimensional. Posteriormente, se utiliza OpenGL para renderizar un objeto 3D, en este caso un cubo, sobre el patrón detectado.

2. Objetivos

- Crear una Aplicación de Realidad Aumentada: Desarrollar una aplicación que combine el mundo real con elementos digitales en tiempo real.
- Utilizar una cámara para encontrar y reconocer ciertos patrones o marcadores en el entorno.
- Mostrar un Objeto 3D: Una vez que el marcador es detectado, la aplicación debe mostrar un objeto 3D, como un cubo, sobre el marcador en la pantalla.
- Ajustar la cámara para que pueda capturar imágenes de manera precisa y corregir cualquier distorsión.

3. Instalación y Configuración del Entorno de Desarrollo

Para comenzar con el proyecto, es esencial configurar el entorno de desarrollo. Esto incluye la instalación de varias bibliotecas y herramientas necesarias:

3.1. Componentes Principales

- **Detección de Patrón**: Utilizando OpenCV, se detecta un patrón de tablero de ajedrez en las imágenes capturadas por la cámara.

- **Estimación de Pose**: Se calcula la posición y orientación del patrón en el espacio 3D para una correcta superposición del objeto virtual.
- **Renderizado 3D**: Utilizando OpenGL, se renderiza un cubo 3D coloreado y texturizado en el centro del patrón detectado.

3.2. Tecnologías utilizadas

- **C++**: Lenguaje de programación utilizado para la implementación del proyecto.
- **OpenCV**: Biblioteca fundamental para el procesamiento de imágenes y detección de marcadores.
- **OpenGL**: API utilizada para crear y renderizar gráficos 3D.
- **GLEW**: Biblioteca que facilita el acceso a extensiones de OpenGL.
- **GLFW**: Biblioteca para la gestión de ventanas y entradas en OpenGL.

El proceso involucra la actualización del sistema, la instalación de dependencias y la configuración de los repositorios de OpenCV.

4. Funcionamiento

1. **Captura de Video**: El programa captura video en tiempo real desde una cámara web.
2. **Detección de Patrón**: Cada frame del video es procesado para detectar el patrón de tablero de ajedrez utilizando funciones de OpenCV.
3. **Estimación de Pose**: Se calcula la pose (posición y orientación) del patrón en el espacio 3D.
4. **Renderizado del Objeto 3D**: Se utiliza OpenGL para renderizar un cubo 3D sobre el patrón detectado.
5. **Visualización**: El resultado final, que combina el video real con el objeto virtual, se muestra en una ventana.

5. Ejecución y Validación del Proyecto

Una vez que el entorno está configurado y el código está listo, se compila y ejecuta la aplicación. Durante esta etapa, verificamos que el cubo 3D se renderiza correctamente sobre el marcador ArUco en tiempo real.

El proceso de validación incluye:

- Compilar el código fuente.
- Ejecutar el programa y observar la renderización del cubo 3D sobre el marcador.

6. Manual de Ejecución

6.1. Requisitos Previos

- **Sistema Operativo**: Linux (Ubuntu recomendado)
- **Compilador C++**: g++ (versión 7.0 o superior)
- **CMake**: Versión 3.10 o superior
- **OpenCV**: Versión 4.0 o superior
- **OpenGL y GLEW**

6.2. Instalación de Dependencias

```
sudo apt-get update
sudo apt-get install build-essential cmake
sudo apt-get install libopencv-dev
sudo apt-get install libglew-dev libgl1-mesa-dev
```

6.3. Compilación del Proyecto

- Navega al directorio del proyecto.
- Crea un directorio de compilación:

```
mkdir build && cd build
```

- Configura el proyecto con CMake:

```
cmake ..
```

- Comopila elproyecto

```
make
```

6.4. Ejecución

1. Asegúrate de tener una cámara web conectada.
2. Desde el directorio 'build', ejecuta:

   ```
   ./ARProject
   ```

3. Muestra un patrón de tablero de ajedrez 6x9 frente a la cámara.
4. Presiona 'ESC' para salir del programa.

1. Asegúrate de tener una cámara web conectada.

2. Desde el directorio 'build', ejecuta:

```
./ARProject
```

3. Muestra un patrón de tablero de ajedrez 6x9 frente a la cámara.

4. Presiona 'ESC' para salir del programa.

7. Ejemplos de Ejecución

7.1. Inicialización del Proyecto

Al iniciar el programa, la cámara debe capturar el entorno y buscar el marcador ArUco. Una vez detectado el marcador, el cubo 3D aparecerá sobre él en la pantalla. A continuación, se muestra el código de ejecución y los resultados esperados:

```
./ARProject
```

7.2. Visualización del Cubo 3D

El cubo 3D debe mantenerse estable sobre el marcador mientras la cámara se mueva, demostrando una integración adecuada entre la detección del marcador y la renderización del objeto 3D. A continuación se presentan algunas imágenes de ejemplo:

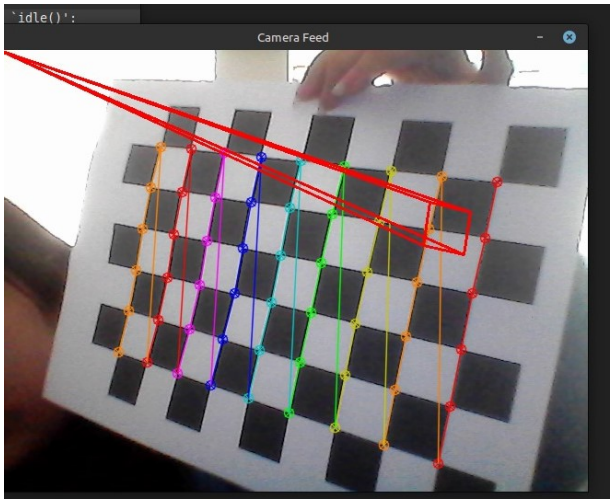

Figura 1: Detección del marcador

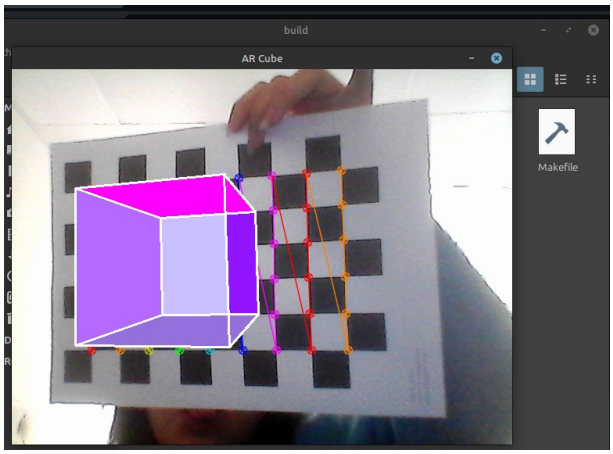

Figura 2: Renderización del cubo 3D sobre el marcador detectado.

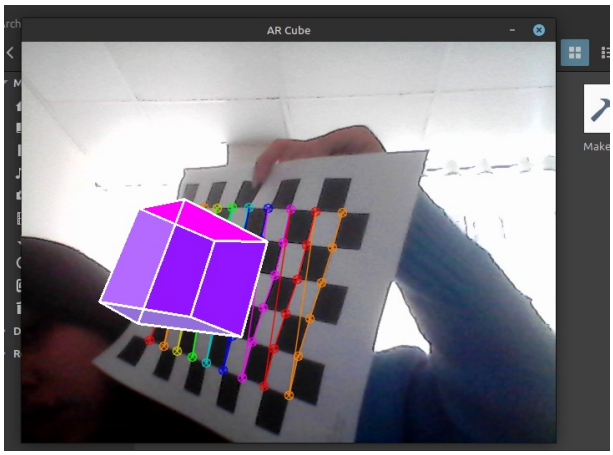

Figura 3: Renderización del cubo 3D sobre el marcador detectado.

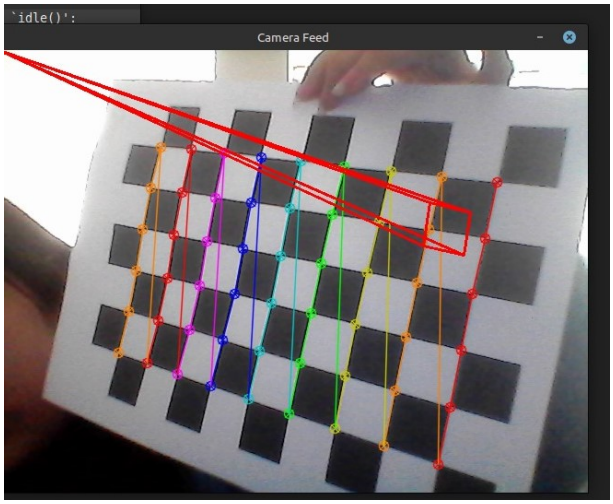

Figura 4: Renderización del cubo 3D sobre el marcador detectado.

8. Código del Proyecto

8.1. Estructura del Proyecto

```
mkdir -p AR_Cube/src
cd AR_Cube
```

8.2. Archivo CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(AR_Cube)

# Especificar el est ndar de C++
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)

# Incluir los directorios
include_directories(include)

# A adir los archivos fuente
file(GLOB SOURCES "src/*.cpp")

# A adir el ejecutable
add_executable(AR_Cube ${SOURCES})

# Enlazar las bibliotecas
find_package(OpenCV REQUIRED)
find_package(glfw3 REQUIRED)
find_package(GLEW REQUIRED)
find_package(OpenGL REQUIRED)

target_link_libraries(AR_Cube ${OpenCV_LIBS} glfw GLEW OpenGL::GL)
```

8.3. Archivo main.cpp

```
#include <opencv2/opencv.hpp>
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <vector>
#include <iostream>

// Funcion para dibujar un cubo
void drawCube() {
    glBegin(GL_QUADS);
    // Cara frontal
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    glVertex3f( 1.0f, -1.0f, 1.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
```

```
19      // Otras caras...
20      glEnd();
21  }
22
23  int main() {
24      // Inicializaci n de OpenCV
25      cv::VideoCapture cap(0); // Abre la c mara
26      if (!cap.isOpened()) {
27          std::cout << "Error al abrir la c mara" << std::endl;
28          return -1;
29      }
30
31      cv::Size patternSize(9, 6); // Tama o del tablero de ajedrez
32      std::vector<cv::Point3f> objPoints;
33      for (int i = 0; i < patternSize.height; i++) {
34          for (int j = 0; j < patternSize.width; j++) {
35              objPoints.push_back(cv::Point3f(j, i, 0));
36          }
37      }
38
39      std::vector<std::vector<cv::Point3f>> objPointsArray;
40      std::vector<std::vector<cv::Point2f>> imgPointsArray;
41
42      cv::Mat cameraMatrix, distCoeffs;
43      std::vector<cv::Mat> rvecs, tvecs;
44
45      int framesForCalibration = 30;
46      int framesCollected = 0;
47
48      // Renderizado y detecci n en tiempo real
49      while (framesCollected < framesForCalibration) {
50          cv::Mat frame;
51          cap >> frame;
52
53          if (frame.empty()) {
54              std::cout << "Error al capturar el frame" << std::endl;
55              break;
56          }
57
58          std::vector<cv::Point2f> corners;
59          bool patternFound = cv::findChessboardCorners(frame, patternSize
        , corners, cv::CALIB_CB_ADAPTIVE_THRESH + cv::
        CALIB_CB_NORMALIZE_IMAGE + cv::CALIB_CB_FAST_CHECK);
60
61          if (patternFound) {
62              cv::cornerSubPix(frame, corners, cv::Size(11,11), cv::Size
        (-1,-1), cv::TermCriteria(cv::TermCriteria::EPS + cv::TermCriteria::
        MAX_ITER, 30, 0.1));
63              imgPointsArray.push_back(corners);
64              objPointsArray.push_back(objPoints);
65              framesCollected++;
66          }
67
68          cv::drawChessboardCorners(frame, patternSize, cv::Mat(corners),
        patternFound);
69          cv::imshow("Frame", frame);
70
```

```
71          if (cv::waitKey(30) >= 0) {
72              break;
73          }
74      }
75
76      cv::destroyAllWindows();
77
78      // Calibraci n de la c mara
79      cv::calibrateCamera(objPointsArray, imgPointsArray, patternSize,
    cameraMatrix, distCoeffs, rvecs, tvecs);
80
81      // Inicializaci n de GLFW y GLEW
82      if (!glfwInit()) {
83          std::cout << "Error al inicializar GLFW" << std::endl;
84          return -1;
85      }
86
87      GLFWwindow* window = glfwCreateWindow(640, 480, "AR Cube", NULL,
    NULL);
88      if (!window) {
89          glfwTerminate();
90          return -1;
91      }
92
93      glfwMakeContextCurrent(window);
94      glewInit();
95
96      while (!glfwWindowShouldClose(window)) {
97          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
98
99          // Renderizado del cubo
100         glMatrixMode(GL_PROJECTION);
101         glLoadIdentity();
102         glm::mat4 projection = glm::perspective(45.0f, 4.0f/3.0f, 0.1f,
    100.0f);
103         glLoadMatrixf(glm::value_ptr(projection));
104
105         glMatrixMode(GL_MODELVIEW);
106         glLoadIdentity();
107         glm::mat4 view = glm::lookAt(glm::vec3(0.0f, 0.0f, 5.0f), glm::
    vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
108         glLoadMatrixf(glm::value_ptr(view));
109
110         drawCube();
111
112         glfwSwapBuffers(window);
113         glfwPollEvents();
114     }
115
116     glfwDestroyWindow(window);
117     glfwTerminate();
118     return 0;
119 }
```

9. Limitaciones o Problemas con el Código Desarrollado

- **Precisión de la Detección del Marcador** La precisión en la detección del marcador puede verse afectada por varios factores, incluyendo la calidad de la imagen, la iluminación del entorno y la distancia entre la cámara y el marcador. Es posible que se requieran ajustes en los parámetros de calibración y en el tamaño del marcador para mejorar la precisión.
- **Rendimiento del Sistema** La renderización en tiempo real de objetos 3D puede ser exigente para sistemas con hardware limitado. La calidad y el rendimiento de la aplicación pueden verse afectados en dispositivos con recursos gráficos inferiores, lo que puede resultar en una experiencia de usuario menos fluida.
- **Calibración de la Cámara** La calibración de la cámara puede ser imprecisa si las imágenes del tablero de ajedrez no se capturan adecuadamente o si el tablero se deforma durante la captura. Es crucial capturar imágenes desde diferentes ángulos y distancias para obtener una calibración precisa.
- **Problemas de Compilación** Pueden surgir problemas durante la compilación si las versiones de las bibliotecas instaladas no son compatibles entre sí o si faltan dependencias. Es importante verificar las versiones de las bibliotecas y las configuraciones de `CMake` para asegurar una compilación exitosa.
- **Limitaciones de OpenCV y OpenGL** Las bibliotecas OpenCV y OpenGL, aunque poderosas, tienen sus propias limitaciones. OpenCV puede no ser capaz de manejar ciertos tipos de distorsión de imagen o condiciones de iluminación extremas, mientras que OpenGL puede tener limitaciones en la renderización de objetos complejos en tiempo real.
- **Estabilidad del Renderizado 3D** La estabilidad del renderizado del objeto 3D sobre el marcador puede verse afectada por el movimiento rápido de la cámara o por cambios abruptos en el entorno. Esto puede resultar en un seguimiento inexacto del marcador y en una visualización inestable del objeto 3D.

10. Conclusión

En este proyecto, hemos explorado la integración de tecnologías de visión por computadora y gráficos 3D para desarrollar una aplicación de realidad aumentada en Linux Mint. A través de la instalación de dependencias esenciales, la configuración de un entorno de desarrollo adecuado, y la implementación de código en C++, hemos conseguido crear una aplicación capaz de detectar marcadores ArUco y superponer objetos 3D en tiempo real.

La calibración de la cámara, un paso crucial en este proceso, nos permitió obtener una mayor precisión en la detección del marcador y la renderización del objeto. A pesar de los desafíos encontrados, como la variabilidad en la calidad de la detección y los requisitos de rendimiento del sistema, el proyecto demuestra efectivamente las capacidades y el potencial de la realidad aumentada.

Este trabajo sienta las bases para futuros desarrollos en aplicaciones de AR más complejas, incluyendo mejoras en la precisión, optimización del rendimiento y la inclusión de funcionalidades adicionales. La experiencia adquirida y los conocimientos obtenidos proporcionan un sólido punto de partida para exploraciones más avanzadas en el campo de la realidad aumentada.